THE MATHEMATICS OF THE BRAIN

FINAL PERFORMANCE REPORT

GRANT # FA9550-08-0129, 3/15/2008 – 3/14/2009

Dr. Victor Eliashberg, Avel Electronics

Dr. Yakov Eliashberg, Stanford University

Contents

1 General methodology

The Mathematics of the Brain (MoB) project is initiated by the DARPA DSO Mathematical Challenge One: *Develop a mathematical theory to build a functional model of the brain that is mathematically consistent and predictive rather than merely biologically inspired.* This section discusses some unique features of the MoB project that distinguish it from other *state-of-the-art* projects aimed at the development of the mathematical (computational) theory of the phenomenon of information processing in the human brain. In what follows we explain the basic principles underlying the general methodology of the MoB project.

1.1 The problem of system integration

The main thrust of the MoB project is to tackle the problem of *system integration* in brain modeling and cognitive modeling. Currently there exists a large and rapidly growing set of different computational models of what can be loosely referred to as the *parts of the brain* and/or the *parts of the brain's behavior.* At the same time, very little attention is paid to the question as to how (and if) such partial models can be integrated in a mathematical theory of the whole brain as an integrated computing system. We consider this question to be crucial for any consistent mathematical theory of the human brain and human cognition.

Anyone who was involved in a nontrivial reverse engineering project can safely say that, in real life, it is seldom (if ever) possible to reverse engineer and understand parts of an unknown integrated computing system (much less, the parts of the behavior of such a system) without having a good initial hypothesis about the basic principles of organization and functioning of the whole system. There is no reason to assume that the situation can be simpler in the case of reverse engineering the brain. After all, the brain is a real working integrated computing system!

It should be mentioned that the critical importance of the problem of system integration in brain modeling was well understood in the early days of cybernetics. Here is a revealing quotation from the neurophysiologist Zopf Jr. (1961) [26]:

"I am intolerant of those who regard the whole of biological data of the phenomena of biological organization and intelligence as not more than a grab bag from which to abstract technological goodies. My intolerance is tempered only by the belief that such casual abstraction may not succeed."

Unfortunately, since the late 1960-s the main trend in brain modeling and cognitive modeling has been toward *system fragmentation* rather than system integration. The concept of the *model for the whole brain* has been taboo[1].

1.2 Three general postulates

Let (W, D, B) be a cognitive model, where W is an external world (environment), D is a set of human-like sensorimotor devices, and B is a computing system simulating the work of the human nervous system – for simplicity, call it the brain. The general structure of the cognitive system (W, D, B) is illustrated in Figure 1. The system (D, B) will be called a *human-like robot* or simply a *robot*, system (W, D) will be called an *external system*.

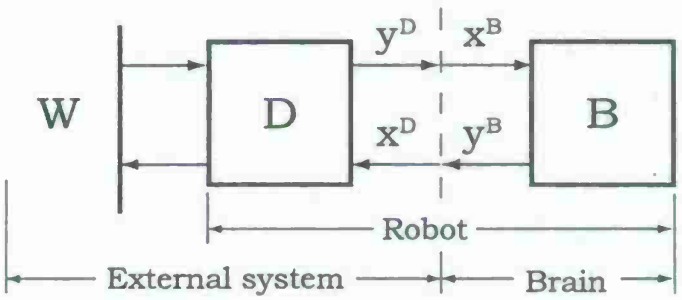

Figure 1: Cognitive system (Robot,World) as a composition of two machines (W,D) and B

At the system-theoretical level, both B and (W, D) can be treated as *abstract machines*, the inputs of B, x^B, being the outputs of (W, D), y^D, and vice versa ($x^D = y^B$).

Let $B(t)$ denote a formal representation of B at time t, and let $t = 0$ correspond to the beginning of learning. Though, in the case of the human brain it is difficult to draw a line between the development and learning, this approach can be thought of as a reasonable initial approximation. In the case of a human-like robot, the beginning of learning can be clearly defined. We proceed from the three general postulates:

P1. **There exists a relatively short formal representation of $B(0)$ as an abstract computing system.**

For the sake of concreteness, we postulate that the size of this representation does not

[1] The big rift between *symbolic* cognitive modeling and *dynamical* neural modeling opened after the famous criticism of the Perceptron [19]. The original spirit of cybernetics has never been restored. The "neural renaissance" that begun in the early 1980-s has not repaired the damage. Arguably, it increased the rift by creating a form of dynamical extremism with a strong *anti-symbolic* mental set. A balanced symbolic/dynamical approach to the problem of the whole brain – like the one represented by this report – became next to impossible to promote.

exceed several megabytes. The exact size is not important for our purpose. What is important is that this size cannot be too big because $B(0)$ is encoded in some form in the human genome. The whole human genom fits into $\approx 700MB$.

P2. **The main part of the formal representation of the trained (educated) brain, B(t), is created in the course of learning.**

Let, for the sake of concreteness, $t = t_{20} = 20years$. Any formal representation of $B(t_{20})$ must be very long (terabytes?). It must include, in some form, a formal representation of an individual experience of a person. That is, an overwhelming part of the formal representation of $B(t_{20})$ is created in the course of learning.

P3. **The essential difference between $B(t)$ and $B(0)$ is in the state of LTM .**

Let $B(t) = (H(t), g(t))$, where $H(t)$ is the representation of the brain hardware, and $g(t)$ is the representation of the brain knowledge (brain software). As a zero-approximation hypotheses we postulate that $H(t) = H(0) = H$.

Remark. This is not literally true in the case of the biological brain. A child grows and becomes an adult. We claim, however, that, functionally, H does not change much. What changes dramatically is the software, g(t).

1.3 Falsification principle

We treat the problem of the human-like robot (D,B) as a *scientific/engineering* problem rather than just an engineering (bionic) problem. The difference is as follows. In the case of the bionic approach, one uses biological data only as a source of inspiration for introducing different engineering/mathematical ideas. In the case of the scientific/engineering approach one relies not only on biological inspiration but also on *biological falsification.*

Falsification principle requires one to pay more attention to the *negative* facts – the facts that contradict to one's ideas — rather than to the *positive* facts – the fact that "confirm" one's ideas. This falsification strategy is very difficult to promote in the case of brain modeling and cognitive modeling. Explaining the nature of the difficulties, the neurophysiologist D. Burns (1958) [4] wrote that, (deling with the brain)

> *it is distressingly easy to find what one is looking for, and remarkably difficult to discern the unsuspected or the unwanted.*

To appreciate the importance of negative facts consider the following gedanken experiment. Imagine that we are testing the hypothesis that a certain object is made of gold. We can find many positive facts "confirming" our hypothesis: the object is heavy, it is yellow, it is shining, etc. However, all these positive facts have no value if we can show that the density of the object's material is, say, $8.9g/cm^3$ – that is, much less than the density of gold, $19.3g/cm^3$. This single negative fact has more power than all positive facts taken together. Our hypothesis is false no matter how many more positive facts we can find.

The power of NO follows from the basic properties of the *if* p *then* q ($p \to q$) statement. If implication q is $false$, then the proposition p is $false$. However, if q is *true*, p can either be *true*

or *false*. That is, false theories can produce true implications. These true implications, however, do not validate a theory. Only false implications have real power. One can say that, in science, YES is just the absence of NO.

The problem of *system integration* outlined in Section 1.1 provides a powerful *falsification test*. Let *a,b,c,...* be some basic properties of the brain as an integrated computing system, and let **A,B,C,...** be the sets of all possible systems with the properties *a,b,c,...*, respectively. If one treats all the above properties as constraints on a single integrated model of the brain, then the search area for the corresponding model is the intersection $\mathbf{A} \bigcap \mathbf{B} \bigcap \mathbf{C}$.... The more properties one considers the smaller becomes the search area. In contrast, if one ignores the requirement of system integration and treats each property separately – just as a biological inspiration for studying the classes of systems with the corresponding "biological" properties – the search area is the union $\mathbf{A} \bigcup \mathbf{B} \bigcup \mathbf{C}$.... Ironically, with the latter approach, the more properties one considers the bigger becomes the search area. The situation is illustrated in Figure 2.

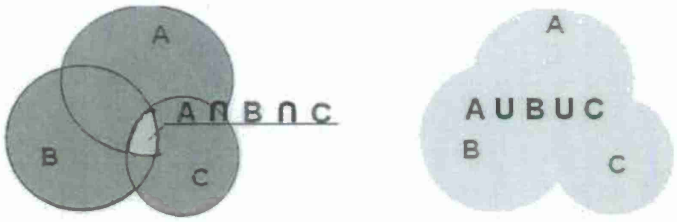

Figure 2: On system integration and falsification.

The essence of the falsification principle is well represented by the famous Sir Arthur Conan Doyle quotation:

> *When you eliminate the impossible, whatever remains, however improbable, must be the truth.*

It is worth mentioning that all successful *scientific* theories rely on *refutation*. In contrast, all *pseudoscientific* theories rely on *confirmation*.

1.4 On basic mechanisms and specific constraints

The general relationship between the mathematical theory of the behavior of the cognitive system, (W,D,B), and the mathematical theory of the brain, B, can be meaningfully compared with the general structure of a traditional physical theory. Let us take the classical electrodynamics as an example of the latter theory. To get a specific methodological metaphor, consider the problem of simulating the behavior of electromagnetic field in the Stanford Linear Accelerator (SLAC). The mathematical model underlying this simulation can be represented as a pair (C,M), where C are the specific external constraints (boundary conditions and sources) representing the design of the SLAC, and M are the Maxwell equations describing the basic laws of electricity and magnetism. The broad predictive power of the classical electrodynamics is a result of an adequate separation of the basic constraints, M, and the specific problem-oriented constraints, C. Without such an adequate separation, each new set of specific constrains would lead to a new phenomenological

theory. We argue that, similarly, it is critically important to adequately divide a consistent mathematical theory of the behavior of system (W,D,B) into "basic constraints", B, and "specific external constraints", (W,D). That is, to develop such a consistent theory, we should concentrate on reverse engineering the basic principles of organization and functioning of B(t) (particularly, B(0)) rather than on the simulation of the specific cognitive phenomena per se. One can safely say that it would be impossible to reverse engineer the Maxwell equations from the analysis of the behavior of electromagnetic field in the SLAC.

2 The concept of E-machine

This report deals with a class of brain models (models of B) called the *E-machines* [6, 7, 8]. In the general case, a *complex E-machine* (CEM) is a hierarchical associative learning system built from several homogeneous associative learning systems called *primitive E-machines* (PEM). A PEM can be thought of as a "higher-level" (procedural) formalization of the intuitive notion of a *context-sensitive associative memory* (CSAM) [9]. This system-level formalization is largely independent of a specific neural implementation. In what follows we first give an abstract description of a PEM and then present a specific example illustrating this concept. The problem of neural implementation of E-machines is discussed in Section 8.

2.1 An abstract description

A primitive E-machine (PEM) is a system $PEM = (\mathbf{X}, \mathbf{Y}, \mathbf{E}, \mathbf{G}, f_y, f_e, f_g)$, where

- $\mathbf{X}$ and $\mathbf{Y}$ are finite sets of symbols called the *input* and the *output* set, respectively;
- $\mathbf{G}$ is the set of states called the *states of encoded (symbolic) long-term memory* (LTM) or the *G-states*. The letter 'G' implies the notion of "synaptic Gain". The G-states represent the symbolic knowledge (software) of an E-machine.
- $\mathbf{E}$ is the set of states, called the *states of dynamical short-term memory* (STM) and *intermediate-term memory* (ITM), or the *E-states*. The letter 'E' implies the notion of "residual Excitation". An E-machine may have several types of E-states representing different temporary attributes (dynamical labels) of the data stored in LTM. The E-states serve as the mechanism for context-dependent dynamic reconfiguration of the knowledge represented by the G-states.
- $f_y : \mathbf{X} \times \mathbf{E} \times \mathbf{G} \rightarrow \mathbf{Y}$ is a function called, interchangeably, the *output procedure*, the *interpretation procedure*, or the *decision making procedure*.[2]
- $f_e : \mathbf{X} \times \mathbf{E} \times \mathbf{G} \rightarrow \mathbf{E}$ is a function called, interchangeably, the *next E-state procedure*, or the *dynamic reconfiguration procedure*.
- $f_g : \mathbf{X} \times \mathbf{E} \times \mathbf{G} \rightarrow \mathbf{G}$ is a function called, interchangeably, the *next G-state procedure*, or the *incremental learning algorithm*.

[2]In general, f_y is a probabilistic procedure, so the symbol "$\rightarrow$" should be interpreted dynamically, as *"compute"*, rather than statically, as *"map"*.

Note. A PEM may have other types of states. For simplicity, these states are not included in the above general description.

The work of a PEM is described in discrete time, ν, as follows:

(1) $$y(\nu) = f_y(x(\nu), e(\nu), g(\nu));$$

(2) $$e(\nu+1) = f_e(x(\nu), e(\nu), g(\nu));$$

(3) $$g(\nu+1) = f_g(x(\nu), e(\nu), g(\nu));$$

with initial states $e(0)$ and $g(0)$,

where $x(\nu) \in \mathbf{X}$, $y(\nu) \in \mathbf{Y}$, $e(\nu) \in \mathbf{E}$, and $g(\nu) \in \mathbf{G}$.

In the next section, we present an explicit example of a PEM. The model is simple enough to be theoretically understandable. At the same time it is sufficiently complex to produce some nontrivial *robopsychological* phenomena. (The term "robopsychology" is borrowed from I. Asimov [2].)

2.2 An example of a primitive E-machine

Figure 3 illustrates the architecture of a simple PEM. The interpretation procedure f_y is divided into four elementary procedures: DECODING, MODULATION, CHOICE, and ENCODING. The model uses a single E-state array, $e(i)$.

The next E-state procedure, f_e, is similar, in this example, to the *fast-charging-slow-discharging* of a capacitor. The model employs a universal learning algorithm, f_g, that simply tape-records input/output associations in the LTM. The recorded association gets an elevated level of residual excitation to produce an effect of *recency*. This is described as the addition to the next E-state procedure.

Variables:

- $\nu \in \{0, 1, ...\}$ is a discrete time (the cycle number).[3]
- $x(:) \doteq x(1), \ldots, x(m)$ is the input vector with m components. In this example, each component is treated as a symbol. That is, only the equal/not equal relationship is defined. We use a special empty symbol, ε, to indicate "no data".
- $wx(:) \doteq wx(1), \ldots, wx(m)$, where $wx(j) > 0$ $(j = 1, ..m)$ is the vector describing the weights of input symbols.
- $gx(:, i) \doteq gx(1, i), \ldots, gx(m, i)$ is the vector stored in the i-th location of the Input LTM (ILTM), where $i \in \{1, ...n\}$.

[3] Treated as real-time cognitive models, E-machines can be thought of as operating with a *psychological time step* Δt on the order of $1 - 10msec$. More complex models of E-machines with multi-step cycles may use several time variables, $\nu 1$, $\nu 2, \ldots$ with different time steps.

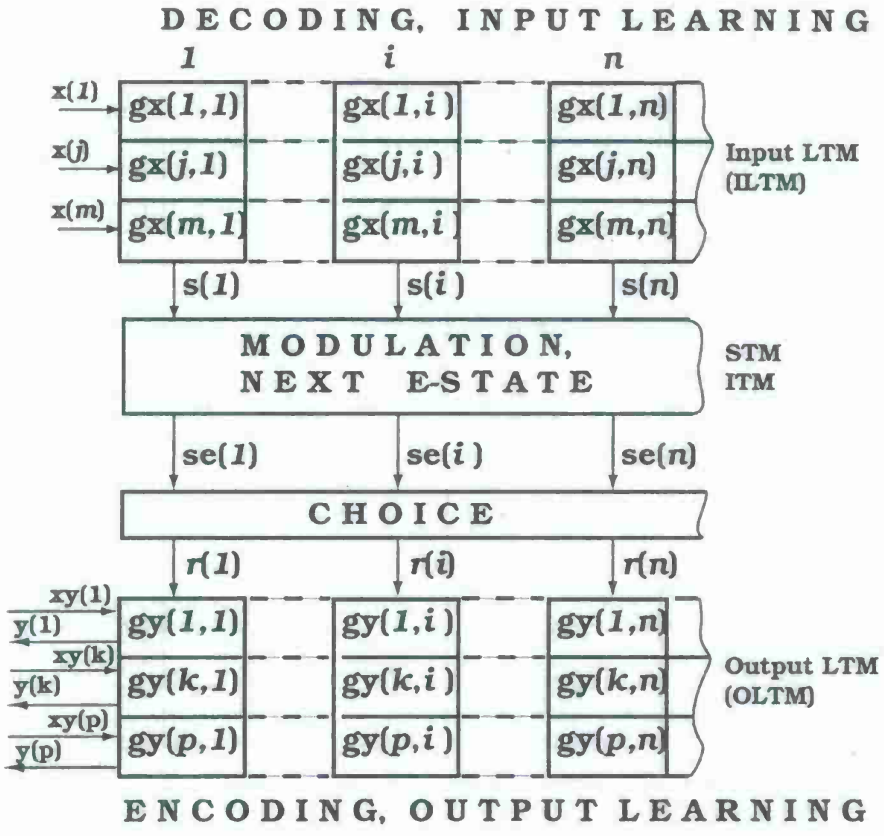

Figure 3: Example of a primitive E-machine

- $s(:) \doteq (s(1), \ldots, s(n))$ is a *similarity array*. In general, $s(i)$ is a nonnegative real number representing a similarity between $x(:)$ and $gx(:, i)$. In this example, we use a very simple criterion of similarity – the number of matching non-empty symbols. Accordingly $s(i) \in \{0, 1, \ldots, m\}$.

- $e(:) \doteq (e(1), \ldots, e(n))$ is an *E-state array*. Variable $e(i)$ is a nonnegative real number that represents the level of *residual excitation* associated with the i-th location of the LTM. In this example, we use a single E-state array. In more complex models, several E-state arrays, $e1(:)$, $e2(:), \ldots$, with different dynamic properties can be used.

- $se(:) \doteq (se(1), \ldots, se(n))$ is *modulated* or *biased similarity array*. In general, $se(i)$ is an element of a real array that describes the similarity affected by the residual excitation.

- $r(:) \doteq (r(1), \ldots, r(n))$ is a *retrieval array*. In general, $r(i)$ is an element of a real array that represents the level of activation of the i-th location of OLTM. In this model, we use a random *winner-take-all* choice, so only one component of this array, $r(iwin)$, corresponding to the winner, $iwin$, is not equal to zero. Formally, in this example we need only the variable $iwin$. The r-array is introduced for the sake of completeness. It does not appear in the following equations. This array is needed in more complex models of primitive E-machines that employ more complex encoding procedures.

- $gy(:,i) = gy(1,i), \ldots, gy(p,i)$ is the vector stored in the i-th location of the Output LTM (OLTM). In this model, components of $gy(:,i)$ are treated as symbols.
- $y(:) = y(1), \ldots, y(p)$ is the output vector retrieved from OLTM. In this model the output is read from the winner location of OLTM. Components of $y(:)$ are treated as symbols.
- $xy(:)$ is the input to the OLTM used for writing output data in this memory.
- *wptr* and *wen* are the auxiliary variables used to describe the tape-recording learning algorithm. They serve as the *write_pointer* and the *write_enable*, respectively.

Parameters:

- $a < 0.5$ is a parameter that determines the modulating effect of $e(i)$ on $s(i)$ that produces biassed similarity $se(i)$.
- $c < 1.0$ is a parameter that determines the rate of decay of $e(i)$. The time constant of decay is $\tau = 1/(1-c)$, so $c = 1 - 1/\tau$.
- m is the number of components in the input vector $x(:)$.
- n is the number of locations in the ILTM and OLTM. In this model, it is the size of all arrays with index i. Namely, $s(i)$,$se(i)$,$e(i)$, $gx(:,i)$, and $gy(:,i)$.
- p is the number of components in the output vector $y(:)$.

Remark. In the experiments discussed in this paper, weights, $wx(:)$, will be treated as parameters. In more complex experiments the pair (x, wx) can be treated as an input. In fact, in this report, we will only need $wx(j) = 1$, for $j = 1, ..m$.

Procedures:

DECODING (computing similarity):

$for\ i = 1 : n$ (this *for* is applied to expressions (1), (2), and (5))

(1) $$s(i) = \sum_{j=1}^{m} wx(j) \cdot T(x(j) = gx(j,i) \neq \varepsilon)$$

where, $T(z) = 1$, if $z = true$, else $T(z) = 0$

MODULATION (computing biased similarity):

(2) $$se(i) = s(i) \cdot (1 + a \cdot e(i))$$

CHOICE (randomly selecting a winner):

(3) $$iwin \ :\in \ \{i : se(i) = max(se) > 0\} \ \doteq \ \mathbf{MSET}$$

where $:\in$ denotes the operation of the random equally probable choice of an element from a set.

ENCODING (retrieving data from OLTM):

(4) $y(:) = gy(:, iwin)$

NEXT E-STATE PROCEDURE (dynamic reconfiguration):

(5) $if\ (s(i) > e(i))\ e(i)(\nu + 1) = s(i)\ ;\ else\ e(i)(\nu + 1) = c \cdot e(i);\ \ end$

where, $e(i)(\nu + 1)$ is the value of $e(i)$ at the next moment $\nu + 1$. For simplicity, we do not write ν in the current values of the variables. That is, $e(i)$ is the same as $e(i)(\nu)$, $s(i)$ is the same as $s(i)(\nu)$, etc.

NEXT G-STATE PROCEDURE (learning)

(6) $if(wen == 1)\ gx(:, wptr)(\nu + 1) = x(:);\ \ gy(:, wptr) = xy(:);$
. $wptr(\nu + 1) = wptr + 1;\ end$

Remark. Expression (6) tape-records input and output vectors in the ILTM and OLTM, respectively, when recording is enabled, $wen = 1$. For simplicity, in this model, we assume that the weights of input symbols, $wx(:)$, do not affect recording. We will always have $wx(j) \geq 1.0$.

ADDITION to the NEXT E-STATE PROCEDURE (the recorded location of LTM, $i = wptr$*, gets initial residual excitation)*

(7) $if(wen == 1)\ e(wptr)(\nu + 1) = s(wptr) = \sum_{j=1}^{m} wx(j) \cdot T(x(j) \neq \varepsilon);\ end$

Remark. The truth function $T(z)$ is defined in expression (1). Expression (7) adds residual excitation to the location of ILTM with $i = wptr$, – the location in which data was just recorded. This happens only if recording is enabled, $wen = 1$. The level of added residual excitation, $e(wptr)$, is equal to the one that would be produced if the input vector $x(:)$ were already recorded in this location of ILTM, that is, as if $gx(:, wptr) = x(:)$. This trick allows one to avoid introducing intermediate steps in the $\nu - th$ cycle.

System of references. The model described above will be referred to as Model (2.2) or PEM (2.2), meaning the model (PEM) described in Section 2.2. Similarly, expression (2.2.6) will mean expression (6) of Section 2.2, etc.

3 Experimental system and the list of tasks

3.1 Experimental system

The structure of the specific cognitive system, (W,D,B), used in this report is shown in Figure 4. The system is intentionally simplified to make it as understandable as possible without losing its ability to illustrate the basic ideas. The ideas are scalable and can be employed in much more complex systems arranged on intuitively similar principles.

The robot, (D,B), interacts with an external world, W, represented by a keyboard and a screen. The screen is divided into squares (only one row of squares is shown). For simplicity, we assume that the robot's eye can scan only one square at a time, call it the *scanned square.* We also assume

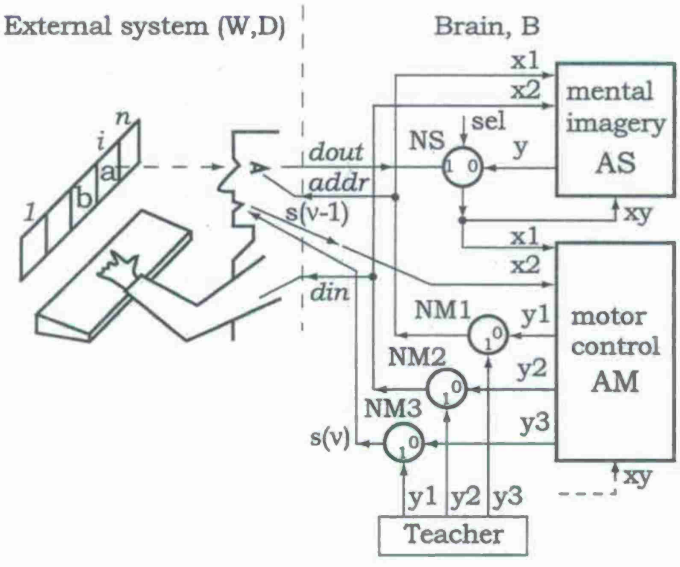

Figure 4: Experimental system

that the system has some eye tracking device (not shown), so when the robot depresses a key the character appears in the scanned square. It is easy to see that the described external system, (W,D), behaves, essentially, as a RAM.

Two *motor* inputs, *addr* and *din*, serve, respectively, as the *address* and *data* inputs of a RAM. The *sensory* (visual) output, *dout*, serves as the *data* output . There is no *control input* similar to the *write-enable* input of a RAM. We can say that system (W,D) is in the *write mode* when there is a *nonempty* motor input, *din*. Otherwise, the system is in the *read mode*. We will formalize this verbal description as the concept of a *generalized RAM* (GRAM).

The robot's brain is divided into four parts:

1. Associative learning system AS responsible for working memory and mental imagery. The goal of this system is to learn to simulate the external system (W,D). In this case, (W,D) works as a RAM.

2. Associative learning system AM responsible for motor control. The goal of this system is to learn to simulate the teacher.

3. Sensory nucleus NS. In this case, NS works as a multiplexer that switches between the output from the eye, *dout*, and the output from AS, $AS.y$.

   Note. We use traditional notation in which $A.y$ means variable y of a unit (object) A.

4. Motor nucleus NM. In this case, NM works as a multiplexer that switches between the output of the teacher, call it $T.y$, and the output of system AM, call it $AM.y$. Both the teacher and AM have three outputs denoted as $y1, y2, y3$.

   Note. We use the same identifiers for the local variables of different units. As mentioned above, we add the name of a unit to the name of a local variable to transform this local

variable into the corresponding global variable of the whole system. That is, $T.y2$ means output $y2$ of unit T; $AM.y1$ means output $y1$ of unit AM, etc.

We will use Model 2.2. with $m = 2$ and $p = 1$ as unit AS. We will use the same model with $m = 2$ and $p = 3$ as unit AM. In the case of unit AS we will need residual excitation $e(:)$, so we will have $AS.a > 0$. In the case of AM, we will set $AM.a = 0$.

3.2 The list of tasks

Here is a list of tasks from the project proposal.

Task 1. A theorem providing the proof that a primitive E-machine with a single E-state function can learn to simulate a RAM buffer without moving symbols in LTM.

Task 2. A theorem providing the proof that a primitive E-machine with a one-step delayed feedback can learn to simulate the finite-state part of any Turing machine.

Task 3. A theorem providing the proof that a system consisting of two primitive E-machines, such as the brain of the robot of Figure 4, can learn to (mentally) simulate any Turing machine – that is, without using an external read/write memory.

Task 4. Develop a C++ program simulating the system of Figure 4.

Task 5. Intermediate performance report

Task 6. Develop a C++ program simulating a neural network performing DECODING.

Task 7. Develop a C++ program simulating a neural network performing random CHOICE.

Task 8. Develop a C++ program simulating a neural network performing ENCODING.

Task 9. Develop a C++ program simulating a neural network performing the NEXT-E-STATE procedure

Task 10. Compile a list of topics for the Phase II of the MoB project.

Task 11. Final performance report.

4 External system as a generalized RAM

In this section and the next one we will go through the following steps:

Step 1. Show that the external system (W,D) of Figure 4 functions essentially as a symbolic RAM buffer. Formalize the notion of such a symbolic RAM buffer by introducing the concept of a generalized RAM (GRAM).

Step 2. Define the problem of learning to simulate a GRAM.

Step 3. Prove the theorem of Task 1.

4.1 The concept of a generalized RAM (GRAM)

In what follows we formalize the verbal description of the work of the external system shown in Figure 4 as the concept of a *generalized RAM* (GRAM). Figure 5 gives two different interpretations of this important concept.

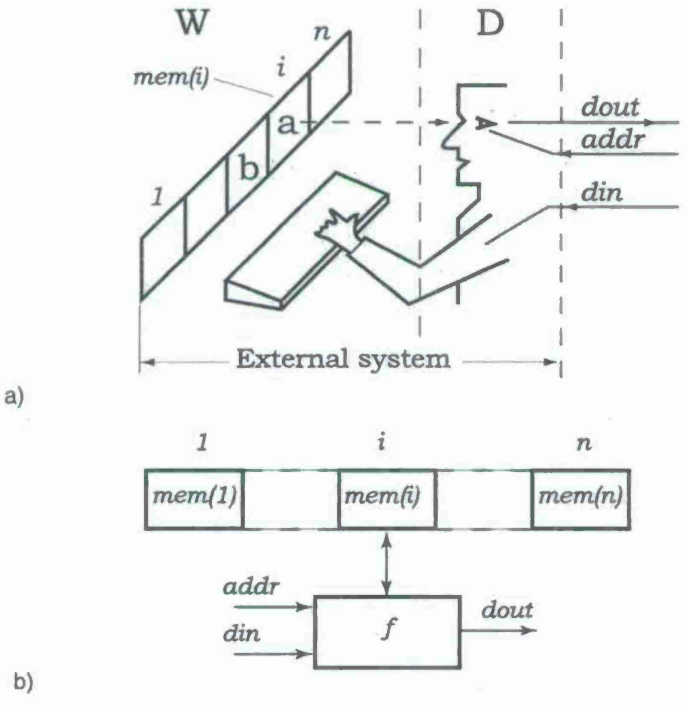

Figure 5: External system as a generalized RAM (GRAM). a) robo-psychological interpretation; b) concept of a generalized RAM (GRAM)

DEFINITION:
A *generalized RAM* (GRAM) is a system $(\mathbf{A}, \mathbf{D}, \mathbf{M}, f)$ (see Figure 5), where

- $\mathbf{A} = \{a_1, \ldots, a_n\}$ is a set of symbols called *address set*. For the sake of simplicity we will assume that $\mathbf{A} = \{1, \ldots, n\}$. In a more general case we would need to define a mapping $\mathbf{A} \leftrightarrow \{1, \ldots, n\}$.

- $\mathbf{D} = \{d_1, \ldots, d_m, \varepsilon\}$ is a set of symbols called *data set*, where ε is the empty symbol meaning "no data".

- $\mathbf{M} = \mathbf{D} \times \ldots \times \mathbf{D} = \mathbf{D}^n$

  is the *set of memory states* represented as memory arrays $(mem(1), \ldots, mem(n))$, where $mem(i) \in \mathbf{D}$ is the data stored in the i-th location of GRAM.

- $f : \mathbf{A} \times \mathbf{D} \times \mathbf{M} \rightarrow \mathbf{D} \times \mathbf{M}$ is a function computing the the output data and the next memory state.

Function f can be represented as the following MATLAB-like program.

(1) $\quad if\ (din == \varepsilon)\ \ dout = mem(addr);$

(2) $\quad else\ \ dout = din;\ mem(addr) = din;\ \ end$

Note. Expressions are numbered independently, starting with (1), in each section. To refer to an expression from a different section we add the section number to the expression number. For example, expression (2.3) is the expression (3) from section 2.

The main difference between a GRAM and a conventional RAM is:

1. In the case of a GRAM, both address and data are treated as symbols: that is, only the *equal/not equal* relation is defined for the elements of $\mathbf{A}$ and $\mathbf{D}$.

2. GRAM is always in the write mode when input data is present, $din \neq \varepsilon$. Only if input data is not present, $din = \varepsilon$, GRAM is in the read mode. With this approach, we do not need a special control input, e.g., *write_enable*, to indicate the write or read mode.

3. In the case of GRAM we do not need $\mathbf{A} = \{1, 2, \ldots, n\}$. We just need to have a mapping $\mathbf{A} \leftrightarrow \{1, 2, \ldots, n\}$. We also do not need input data symbols to be the same as output data symbols. For simplicity, we used the same data set $\mathbf{D}$ for *din* and *dout* rather than two different sets, say, $\mathbf{D}^{in}$ and $\mathbf{D}^{out}$. What we need is a mapping, $\mathbf{D}^{in} \leftrightarrow \mathbf{D}^{out}$. Any mappings $\mathbf{A} \leftrightarrow \{1, 2, \ldots, n\}$ and $\mathbf{D}^{in} \leftrightarrow \mathbf{D}^{out}$ can be learned, so specifying specific mappings makes no difference. We do not even need one-to-one mappings: several symbols form $\mathbf{A}$ could be associated with the same symbol from $\{1, 2, \ldots, n\}$, and several symbols from $\mathbf{D}^{in}$ with the same symbol from $\mathbf{D}^{out}$.

4.2 Experiment with a GRAM

To get used to the notion of GRAM, it is helpful to follow the experiment with a GRAM shown in Figure 6, where $\mathbf{A} = \{1, 2\}$, and $\mathbf{D} = \{a, b, \varepsilon\}$. The three-row table in the upper part of the figure displays the input/output sequence of GRAM as a function of discrete time $\nu = 0, 1, ..9$. The $din = \varepsilon$ entries are shown as blank squares. The two-row table in the lower part of the figure displays the contents of the two memory locations, $mem(1), mem(2)$ as functions of ν. The $mem(i) = \varepsilon$ entries are shown as blank squares. Let us follow the 10 cycles of the experiment:
$\nu = 0$:
Memory is empty: $mem(:) = (mem(1), mem(2)) = (\varepsilon, \varepsilon)$. Input: $(addr, din) = (1, a)$, produces output: $dout = a$, and writes a into location 1: $mem(1) = a$.
$\nu = 1$:
Memory state is: $mem(:) = (a, \varepsilon)$. Input: $(addr, din) = (2, a)$, produces output: $dout = a$, and writes a into location 2: $mem(2) = a$.
$\nu = 2$:
Memory state is: $mem(:) = (a, a)$. Input: $(addr, din) = (1, b)$, produces output: $dout = b$, and writes b into location 1: $mem(1) = b$.
$\nu = 3$:

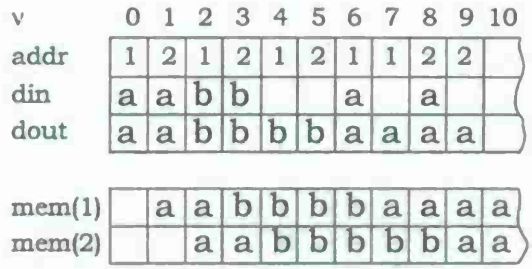

Figure 6: Experiment with a GRAM. $\mathbf{A} = \{1, 2\}$, $\mathbf{D} = \{a, b, \varepsilon\}$. Empty symbol ε is shown as a blank square.

Memory state is: $mem(:) = (b, a)$. Input: $(addr, din) = (2, b)$, produces output: $dout = b$, and writes b into location 2: $mem(2) = b$.
$\nu = 4$:
Memory state is: $mem(:) = (b, b)$. Input: $(addr, din) = (1, \varepsilon)$, reads data from location 1: $dout = mem(1) = b$. Memory state does not change.
$\nu = 5$:
Memory state is: $mem(:) = (b, b)$. Input: $(addr, din) = (2, \varepsilon)$, reads data from location 2: $dout = mem(2) = b$. Memory state does not change.
$\nu = 6$:
Memory state is: $mem(:) = (b, b)$. Input: $(addr, din) = (1, a)$, produces output: $dout = a$, and writes a into location 1: $mem(1) = a$.
$\nu = 7$:
Memory state is: $mem(:) = (a, b)$. Input: $(addr, din) = (1, \varepsilon)$, reads data from location 1: $dout = mem(1) = a$. Memory state does not change.
$\nu = 8$:
Memory state is: $mem(:) = (a, b)$. Input: $(addr, din) = (2, a)$, produces output: $dout = a$, and writes a into location 1: $mem(2) = a$.
$\nu = 9$:
Memory state is: $mem(:) = (a, a)$. Input: $(addr, din) = (2, \varepsilon)$, reads data from location 2: $dout = mem(2) = a$. Memory state does not change.

4.3 Fixed rules and variable rules

Analyzing the three-row input/output table shown in the upper part of Figure 6 we can discover two types of $x \rightarrow y$ rules, where $x = (addr, din)$ and $y = dout$:

1. Rules of the type $(addr, din \neq \varepsilon) \rightarrow dout$, call them *fixed rules*. In this specific example, the fixed rules are: $(1, a) \rightarrow a$, $(2, a) \rightarrow a$, $(1, b) \rightarrow b$, and $(2, b) \rightarrow b$. There are $m \cdot n = 2 \cdot 2 = 4$ such rules. Fixed rules can be easily extracted from the shown xy-sequence by different learning algorithms.

2. Rules of the type $(addr, din = \varepsilon) \rightarrow dout$, call them *variable rules*. In this example, the variable rules are: $(1, \varepsilon) \rightarrow dout$, and $(2, \varepsilon) \rightarrow dout$. There are n variable rules. The output part, $dout$, of a variable rule depends on the most recently executed fixed rule with the same

address. For example, the output of rule $(1, \varepsilon) \rightarrow dout$ at $\nu = 7$ is $dout = a$, because the most recently executed fixed rule with $addr = 1$ is rule $(1, a) \rightarrow a$ at $\nu = 6$. Variable rules cannot be correctly executed by a learning system that does not save information about the most recently executed fixed rules.

It is useful to view fixed rules as a tool for assigning the right parts of variable rules. With this approach, we can say that the *meaning* or *value* of an address symbol in a variable rule depends on the most recent assignment. In the discussed example, each address symbol from $\mathbf{A} = \{1, 2\}$ can be assigned any of two meanings (data values) from $\mathbf{D} - \{\varepsilon\} = \{a, b\}$.

5 Problem of learning to simulate a GRAM

5.1 Defining experiments of training and examination

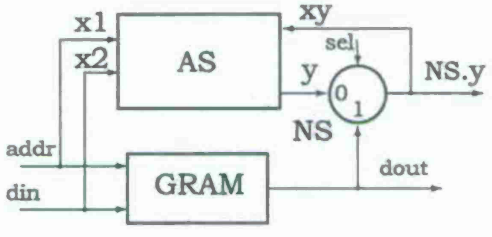

Figure 7: Experimental setup for learning to simulate a GRAM

The problem of *learning to simulate a GRAM* can be defined as a traditional supervised learning problem illustrated in Figure 7. Figure 7 is just a redrawn part of Figure 4 in which external system (W,D) is replaced by the GRAM. We keep the name AS for the learning system. We could use any other name since the problem of learning to simulate a GRAM is a problem independent of Figure 4. Inputs $AS.x1$ and $AS.x2$ are connected to the *addr* and *din* inputs of the GRAM, respectively. Input $AS.xy$ – that can be thought of as the *desired output* – is connected to the output of the multiplexer $NS.y$. GRAM plays the role of the teacher for system AS.

To prove the theorem of Task 1 we assume that system AS is replaced by Model 2.2 with m=2, and p=1. That is $x(:) = x(1), x(2) \in \mathbf{A} \times \mathbf{D}$ and $y(:) = y(1) \in \mathbf{D}$. It is convenient to divide an experiment with the system of Figure 7 into two stages: *training* and *examination=test*:

Training. We assume that training starts at $\nu = \nu 0$ and lasts until $\nu = \nu 1 - 1$, where $\nu 1$ is the first moment of examination. We also assume that the following conditions are satisfied during training:

1. $sel = 1$, and, accordingly, $xy = dout$.
2. At the beginning of training Model 2.2 has empty ILTM and OLTM and zero residual excitation. That is, for $i = 1 : n$ $gx(1, i)(\nu 0) = gx(2, i)(\nu 0) = \varepsilon$; $gy(1, i)(\nu 0) = \varepsilon$, and $wptr(\nu 0) = 1$.

3. Learning is enabled that is, $wen = 1$. We assume that and AS has sufficiently big LTM to record all training sequence. That is $AS.n \geq \nu 1 - \nu 0$, where $AS.n$ is the number of locations of LTM of Model 2.2.

4. Training continues until each pair $(addr, din)$ from $\mathbf{A} \times (\mathbf{D} - \{\varepsilon\})$ is presented at least once. This means that each fixed rule of the simulated GRAM was demonstrated to AS at least once. Let m be the number of nonempty data symbols of GRAM. The number of fixed rules, $nfixed = n \cdot m$, where n is the number of address symbols. That is, $\nu 1 - \nu 0 \geq nfixed$.

Examination. At this stage, $sel = 0$, and $AS.xy = NS.y$. For simplicity, we assume that AS does not learn during this stage, that is, $wen = 0$. This requirement is not necessary. If $AS.n$ is big enough, AS can continue self-learning during this stage. It will not affect the ability of AS to simulate GRAM.

5.2 Proving the theorem of Task 1

Theorem 1. *The PEM described in Section 2.2 (Model 2.2) used as system AS in the experimental setup of Figure 7 can learn to simulate a GRAM from a sample of the GRAM's behavior of the length* $\geq n \cdot m$.

Proof. We begin with a verbal explanation of the *effect of working memory* . The effect is produced by decaying residual excitation, $e(i)$, associated with locations of LTM.

Let the i-th locations of the LTM (ILTM and OLTM) contain the record of the following fixed rule: $gx(1,i) = a, gx(2,i) = d, gy(1,i) = d$, where $a \in \mathbf{A}$ and $d \in \mathbf{D} - \{\varepsilon\}$. The residual excitation $e(i)$ associated with this location can reach the maximum possible level, $emax = 2$, in two situations:

1. At the next moment after the rule was recorded in the i-th location. This is the result of expression (7) from Section (2.2) (expression (2.2.7)) which sets residual excitation at the moment of recording.

2. If the rule is already recorded and input is $x(1)(\nu) = a$, and $x(2)(\nu) = d$. In this case, $e(i)(\nu + 1) = 2$ is a result of expressions (2.2.1) and (2.2.5).

Once the $e(i) = emax$ is set, we can say that the rule (a, d, d) is *placed in working memory.* The reason for this statement is that, if we send input $x(1) = a$ $x(2) = \varepsilon$ the output $y(1)$ will be retrieved from one of the locations of LTM with the highest level of residual excitation among locations for which $gx(1,i) = a$. Locations for which $gx(1,i) \neq a$ will have $s(i) = 0$ due to expression (2.2.1). Accordingly, due to expression (2.2.2), for these locations $se(i) = 0$ independently of the level of residual excitation. This effect of executing the most recent rule placed in working memory will last until $e(i)$ decays below a certain level, $eloss$. For this model $eloss = 1$ and the modulating coefficient, a, in expression (2.2.2) must be less than $1/emax = .5$. Due to expression (2.2.5), the time of decay of $e(i)$ from $emax$ to $eloss$ is

$$(1) \quad tmax = \ln(eloss/emax)/\ln(c) = -\ln(emax/eloss) \cdot \ln(1 - 1/\tau) > \tau \cdot ln(emax/eloss)$$

where $c = 1 - 1/\tau$, and, for this model, $emax/eloss = 2$. We can transform this qualitative explanation into a rigorous proof by verifying the following statements:

S1. At $\nu = \nu 1$ (the beginning of examination) LTM of AS contains at least once any fixed rule $(addr, din, dout) \in \mathbf{A} \times \mathbf{D}' \times \mathbf{D}'$, where $\mathbf{D}' = \mathbf{D} - \{\varepsilon\}$. Formally,

(2) $\quad \forall (addr, din, dout) \in \mathbf{A} \times \mathbf{D}' \times \mathbf{D}' \; \exists i (gx(1,i), gx(2,i), gy(1,i)) = (addr, din, dout))$

This statement follows from expression (2.2.6) and the definition of the experiment of training. During training $wen = 1$, so the training sequence containing all fixed rules is tape-recorded in LTM.

S2. At $\nu = \nu 1$ all location of LTM containing fixed rules have $e(i) > 0$ due to expression (2.2.7). Let $tmax > \nu 2 - \nu 0$. From expression (1) we find that to guarantee that $e(i) = emax = 2$ will not decay below $eloss$ during the entire experiment of training and examination it is sufficient to have $tmax \geq \nu 2 - \nu 0$. This condition is satisfied if

(3) $\quad \tau \geq \tau min \doteq (\nu 2 - \nu 0)/ln(eloss/emax) = (\nu 2 - \nu 0)/ln 2$

S3. Suppose in the course of examination we test AS only in the read mode. That is, we send only inputs $addr, din$ with $din = \varepsilon$. Then $y(1)$ will be retrieved from a location ird for which $gx(1, ird) = addr$ and $e(ird)$ has the highest level among all locations with $gx(1,i) = addr$. If $\tau \geq \tau min$, it is guarantied that $e(ird) > eloss$, so $y(1)$ will be retrieved from the location in which a certain data, $din \neq \varepsilon$ was recorded most recently and, therefore, $y(1) = din$. This is exactly how GRAM would react to this input.

S4. Let νw and νr, where $\nu r > \nu w$ be some "write" and "read" moments in the course of examination, such that $x(1)(\nu w) = addr$, $x(2)(\nu w) = din \neq \varepsilon$ and $x(1)(\nu r) = addr$, $x(2)(\nu r) = \varepsilon$. Let there be no other input between νw and νr with the same $x(1)$ and different $x(2) \neq \varepsilon$. Due to expression (2.2.5) $e(i)(\nu w + 1) = emax = 2$ for all $i \in \{i : gx(1,i) = addr, gx(2,i) = din\}$. Accordingly, due to what was said in **S3**, $y(1)(\nu r) = din$. Again, AS reacts exactly as GRAM.

This proves the theorem, since all other possible cases of "writing to working memory" and reading from this memory can be reduced to **S3** and **S4**. □

Why, in this model should the data loss level, eloss, be set at 1? The reason for this is that the residual excitation $e(i) = 1$ can be created in a wrong location by $x(2) = din \neq \varepsilon$. Accordingly, only the residual excitation higher than 1 can be relied upon. The $e(i) > 1$ can only be the result of decay from level emax=2. The latter level can be set only by the input $x(1) = addr$, $x(2) = din \neq \varepsilon$. This input corresponds to writing non-empty data, $din \neq \varepsilon$, to an address, $addr$, of GRAM.

5.3 Why traditional learning systems cannot learn to simulate GRAM

We are going to explain why many traditional learning systems cannot learn to simulate a GRAM. Intuitively, the obvious cause of this limitation is the loss of information about the relative order

of observed input events. To formalize this intuition, consider a system M the work of which is represented in the following general form:

$$y(\nu) = f(x(\nu), x(\nu-1), \ldots, x(0), y(\nu-1), \ldots, y(0)) \tag{4}$$

where $x(\nu) \in \mathbf{X}$, and $y(\nu) \in \mathbf{Y}$).

The work of all learning systems we are aware of can be represented in this general form. Usually, the I/O sequence, $x(\nu-1), \ldots, x(0), y(\nu-1), \ldots, y(0)$ is transformed into a state of LTM by a learning algorithm. The output, $y(\nu)$, is then computed from the current input, $x(\nu)$, and the current state of LTM. Some systems also use different kinds of STM. (For example, in the case of Model 2.2, the STM was represented by the E-state array $e(:)$.) In the general form (4), we do not need the concepts of LTM, and STM, and the concept of a learning algorithm. Theoretically, a learning algorithm that loses information can only reduce the possibilities of system M. Our goal is to show that this is happening in many traditional learning systems.

Definition. We will say that system M *loses information about the order of events in the input sequence* if there exist ν_1 and ν_2, where $\nu > \nu_2 > \nu_1 \geq 0$, such that for any $a, b \in \mathbf{X}$ with $a \neq b$ the output of system M at the moment ν, $y(\nu)$, is the same for $x(\nu_1) = a$, $x(\nu_2) = b$, and $x(\nu_1) = b$, $x(\nu_2) = a$. That is if inputs at ν_1 and ν_2 change places.

Theorem 2. *No system M satisfying the above definition can learn to simulate GRAM.*

Proof. We prove this theorem by contradiction. It is sufficient to use a specific example of GRAM with $\mathbf{A} = \{1, 2\}$, and $\mathbf{D} = \{a, b, \varepsilon\}$ discussed in Section 4.2. It is easy to see that if M cannot simulate this GRAM it cannot simulate a GRAM with bigger address a nd data sets, because a bigger GRAM can always simulate a smaller GRAM. Suppose M satisfies the above definition and nevertheless can simulate the specified GRAM. Let us make the following test. At $\nu = \nu_1$, we send input $x(\nu_1) = (addr, din)(\nu_1) = (1, a)$, and at $\nu = \nu_2$, we send input $x(\nu_2) = (addr, din)(\nu_2) = (1, b)$. We make sure that no other input between ν_1 and ν_2 has $addr = 1$, and $din \neq \varepsilon$. At the moment ν we send input $x(\nu) = (addr, din)(\nu) = (1, \varepsilon)$. According to the assumption that M simulates the described GRAM the output $y(\nu)$ must be equal to b, since $din = b$ was written to $addr = 1$ last. Let us change the order of inputs at ν_1 and ν_2. Now $x(\nu_1) = (addr, din)(\nu_1) = (1, b)$, and $x(\nu_2) = (addr, din)(\nu_2) = (1, a)$. According to the definition of M the output $y(\nu) = dout(\nu)$ must be the same as before, that is, it must be $y(\nu) = dout(\nu) = b$. However, the output of the GRAM is now $dout(\nu) = a$ since $din = a$ was written to $addr = 1$ last. This contradiction proves the theorem. □

Remark. Using computer simulation, we tested a number of traditional learning systems against the *loss-of-information-about-order* criterion specified by the above definition. We found that all tested systems satisfy this criterion, and hence they cannot learn to simulate GRAM. For example, we claim that this result holds for all systems with distributed associative memory and all systems that employ various types of *gradient-descent* and *statistical-optimization* learning algorithms. We leave it to the researchers who study these types of learning systems to rigorously prove this interesting *negative* result.

Recall what was said in Section 1.3. The GRAM simulation problem offers an important falsification test – much more difficult than the famous XOR problem.

6 Learning to simulate finite-state machines (Task 2)

6.1 The main steps for solving Task 2

We proceed in the following steps:

Step 1. Define the concepts of a deterministic and probabilistic combinatorial machines and the notion of behavior of type 4.

Step 2. Prove that Model 2.2 with $a = 0$, $m = 1$, $p = 1$, and sufficiently big LTM (n as big as needed) can be trained to simulate, in principle, any probabilistic combinatorial machine with rational probabilities.

Step 3. Define the concepts of a deterministic and probabilistic finite-state machines and the notion of behavior of type 3. Show that any finite-state machine can be implemented as a combinatorial machine with a one-step delayed feedback.

Step 4. Define the problem of learning to simulate a behavior of type 3.

Step 5. Prove the theorem of Task 2 using Model 2.2 with $a = 0$, $m = 2$, $p = 3$, and n big enough to store the program of the simulated Turing machine.

6.2 The concept of a combinatorial machine: behavior of type 4

Definitions:

1. A *deterministic combinatorial machine* is a system M=($\mathbf{X}$,$\mathbf{Y}$,f), where

   - $\mathbf{X}$ is a finite nonempty set of input symbols called the *input set* or *input alphabet.*
   - $\mathbf{Y}$ is a finite nonempty set of output symbols called the *output set* or *output alphabet.*
   - $f : \mathbf{X} \to \mathbf{Y}$ is a function called *output function.*

   The work of M is described in discrete time ν as follows :

   (1) $$y(\nu) = f(x(\nu));$$

   where, $y(\nu), b \in \mathbf{Y}$, and $x(\nu), a \in \mathbf{X}$.

2. A *probabilistic combinatorial machine* is a system M_p=($\mathbf{X}$,$\mathbf{Y}$,f_p), where

   - $\mathbf{X}$ and $\mathbf{Y}$ are the same as before.
   - $f_p : \mathbf{X} \times \mathbf{Y} \to [0, 1]$ is the *function of output conditional probabilities* satisfying the condition: for all $x \in \mathbf{X}$ $\sum_{(y \in \mathbf{Y})} f_p(x, y) = 1$.

   The work of M_p is described in discrete time ν as follows :

   (2) $$P(\ y(\nu) = b \mid x(\nu) = a\ ) = f_p(a, b);$$

where, $P(\ B \mid A\ )$ is the conditional probability of event B given event A,
$y(\nu), b \in \mathbf{Y}$, and $x(\nu), a \in \mathbf{X}$.

Note. A deterministic combinatorial machine is an extreme case of a probabilistic combinatorial machine with $f_p(a,b) \in \{0,1\}$.

3. A behavior that can be represented in terms of a combinatorial machine will be called a *behavior of type 4*. In psychological terms, this behavior can be interpreted as the classical $S \rightarrow R$ (stimulus $\rightarrow$ response) behavior.

   Note. We use a classification of the general types of behavior and the levels of computing power similar to Chomsky's hierarchy of formal languages [5]. We added type 4 to refer to combinatorial machines. As is known, types 3, 2, 1, and 0 correspond, respectively, to finite-state machines, context-free grammars or push-down automata, context-sensitive grammars, and Turing machines. The lower the type number, the higher is the level of computing power needed to simulate the corresponding type of behavior. In what follows, we are interested only in types 4, 3, and 0.

6.3 Two examples of combinatorial machines

To get used to the concepts of a deterministic and probabilistic combinatorial machine it is helpful to understand in details the two simple examples shown in Figure 8 a) and b).

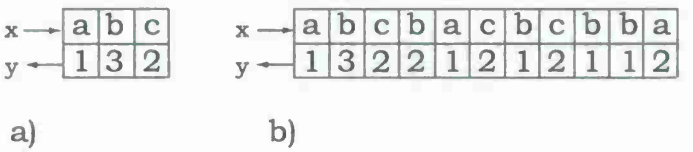

Figure 8: Tables of associations corresponding to a) deterministic, and b) probabilistic combinatorial machines.

a) Example of a deterministic combinatorial machine:

Let $M = (\mathbf{X}, \mathbf{Y}, f)$ be a deterministic combinatorial machine with the input alphabet $\mathbf{X} = \{a, b, c\}$, the output alphabet $\mathbf{Y} = \{1, 2, 3\}$ and the function $f : \mathbf{X} \rightarrow \mathbf{Y}$ represented as the set of pairs, $f = \{(a,1), (b,3), (c,2)\}$. It is useful to visualize f as the table, call it the *table of associations*, shown in Figure 8 a.

At each moment ν, the work of machine M can be described with the use of the following algorithm that *interprets* the table of associations. We will refer to such an algorithm as an *interpretation procedure*:

1. Take an input symbol $x(\nu) \in \mathbf{X}$. For example, let $x(\nu) = b$.
2. Find the column, i with the matching symbol in the upper row. In this case, column $i = 2$ has symbol b in the upper raw.

3. Take the symbol from the second raw of the $i-th$ column and send it to the output. In this case, $i = 2$, and $y(\nu) = 3$.

b) Example of a probabilistic combinatorial machine:

Let $M_p = (\mathbf{X}, \mathbf{Y}, f_p)$ be a probabilistic combinatorial machine with the input alphabet $\mathbf{X} = \{a, b, c\}$, the output alphabet $\mathbf{Y} = \{1, 2, 3\}$ and the function of output conditional probabilities, $f_p : \mathbf{X} \times \mathbf{Y} \to [0, 1]$, represented by the table shown in Figure 8 b. Let $\mathbf{I}(x)$ be the set of columns with symbol x in the first row, and let $\mathbf{I}(x, y)$ be the set of columns with symbol x in the first row and symbol y in the second raw. The conditional probability is determined by the following expression:

$$P(\ y(\nu) = b \mid x(\nu) = a\ ) = f_p(a, b) = |\mathbf{I}(a, b)|/|\mathbf{I}(a)|; \tag{3}$$

Table of Figure 8 b gives the following values for f_p:

$f_p(a, 1) = |\mathbf{I}(a, 1)|/|\mathbf{I}(a)| = 2/3$, where $\mathbf{I}(a) = \{1, 5, 11\}$, and $\mathbf{I}(a, 1) = \{1, 5\}$.

$f_p(a, 2) = |\mathbf{I}(a, 2)|/|\mathbf{I}(a)| = 1/3$, where $\mathbf{I}(a) = \{1, 5, 11\}$, and $\mathbf{I}(a, 2) = \{11\}$.

$f_p(a, c) = |\mathbf{I}(a, 1)|/|\mathbf{I}(a)| = 0/3$, where $\mathbf{I}(a) = \{1, 5, 11\}$, and $\mathbf{I}(a, c) = \{\}$.

$f_p(b, 1) = |\mathbf{I}(b, 1)|/|\mathbf{I}(b)| = 3/5$, where $\mathbf{I}(b) = \{2, 4, 7, 9, 10\}$, and $\mathbf{I}(b, 1) = \{7, 9, 10\}$.

$f_p(b, 2) = |\mathbf{I}(a, 2)|/|\mathbf{I}(b)| = 1/3$, where $\mathbf{I}(b) = \{2, 4, 7, 9, 10\}$, and $\mathbf{I}(b, 2) = \{4\}$.

$f_p(b, 3) = |\mathbf{I}(a, 3)|/|\mathbf{I}(b)| = 1/3$, where $\mathbf{I}(b) = \{2, 4, 7, 9, 10\}$, and $\mathbf{I}(b, 3) = \{2\}$.

$f_p(c, 1) = |\mathbf{I}(c, 1)|/|\mathbf{I}(c)| = 0/3$, where $\mathbf{I}(c) = \{3, 6, 8\}$, and $\mathbf{I}(c, 1) = \{\}$.

$f_p(c, 2) = |\mathbf{I}(c, 2)|/|\mathbf{I}(c)| = 3/3$, where $\mathbf{I}(c) = \{3, 6, 8\}$, and $\mathbf{I}(c, 2) = \{3, 6, 8\}$.

$f_p(c, 3) = |\mathbf{I}(c, 3)|/|\mathbf{I}(c)| = 0/3$, where $\mathbf{I}(c) = \{3, 6, 8\}$, and $\mathbf{I}(c, 3) = \{\}$.

The following probabilistic interpretation procedure simulates the described machine:

1. Take an input symbol $x(\nu) \in \mathbf{X}$. For example, let $x(\nu) = b$.
2. Find the set $\mathbf{I}(b) = \{2, 4, 7, 9, 10\}$. Randomly select an element, i_{rd} from $\mathbf{I}(b)$. Let $i_{rd} = 7$.
3. Read the symbol from the second raw of the $i_{rd} = 7$ column and send it to the output. In this case, $y(\nu) = 1$.

DEFINITION. The columns of the table of associations are (interchangeably) called *associations=rules=productions=commands* of a combinatorial machine.

Unlike the case of GRAM, the behavior of a combinatorial machine is fixed and is completely represented by its *rules=commands*. In the case of a deterministic combinatorial machine, the number of commands is $N_d^c = |\mathbf{X}|$. In the case of a probabilistic combinatorial machine, the number of commands is $N_p^c = |\mathbf{X}| \cdot |\mathbf{Y}|$. Note that, in the case of a probabilistic combinatorial machine, the size of the table of associations must be larger than N_p^c to represent the probabilities. Only the rational probabilities, m/n, – where m is a nonnegative integer, and n is a positive integer – can be represented in this way.

6.4 Learning behavior of type 4

In what follows we use experimental setup shown in Figure 9. We assume that PEM is organized as Model 2.2 with $a = 0$, $m = 1$, and $p = 1$. Since $a = 0$, the STM of the model is turned off and the values of $e(:)$ and c make no difference. As in Section 5.1, we divide an experiment into two stages: training and examination.

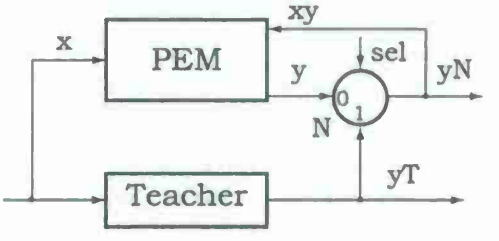

Figure 9: Experimental setup for training a PEM to simulate a combinatorial machine.

Training. Training starts at $\nu = \nu 0$ and lasts until $\nu = \nu 1 - 1$, where $\nu 1$ is the first moment of examination. The following conditions are satisfied:

1. $sel = 1$, and, accordingly, $xy = yT$.
2. At the beginning of training, LTM is empty. That is, for $i = 1 : n$ $gx(1, i)(\nu 0) == \varepsilon$; $gy(1, i)(\nu 0) = \varepsilon$, and $wptr(\nu 0) = 1$.
3. Learning is enabled: $wen = 1$. We assume that the LTM is big enough to store all training sequence. That is $n \geq \nu 1 - \nu 0$.

Examination. At this stage, $sel = 0$, and $xy = y$. For simplicity, we assume that PEM does not learn during this stage, that is, $wen = 0$. This requirement is not necessary. If n is big enough, PEM could continue recording data in its LTM.

Theorem 3. *Let $M = (\boldsymbol{X}, \boldsymbol{Y}, f)$ be a deterministic combinatorial machine. Let each command of these machine be demonstrated at least once during training. At the stage of examination PEM, specified above, will correctly simulate M.*

Proof. Model 2.2 with $a = 0$, $m = 1$, and $p = 1$ performs the probabilistic interpretation procedure described in Section 6.3, where the table of associations is represented as $gx(1, :), gy(1, :)$. Namely,

1. Expressions (2.2.1) and (2.2.3) select the set of matching locations in $gx(1, :)$ (DECODING, and CHOICE). Expression (2.2.2) has no effect: $se(i) = s(i)$ because $a = 0$.
2. Expression (2.2.4) reads the symbol from the selected location of $gy(1, :)$.

Due to the specified conditions of training, at the beginning of examination, $\nu = \nu 1$, the table $(gx(1, :), gy(1, :))(\nu 1)$ includes at least once each command of machine M. This proves the theorem. □

Note. To accomplish Task 2, it is sufficient to prove the above theorem for the case of a deterministic combinatorial machine. However, it is interesting to mention that if: a) M is a probabilistic combinatorial machine, b) training lasts long enough, and c) symbols from $\mathbf{X}$ are presented with equal probabilities, then the resulting table of associations $((gx(1,:), gy(1,:))(\nu 1)$ will correctly approximate the conditional probabilities of M. The longer the training the better will be the approximation.

6.5 The concept of a finite state machine: behavior of type 3

Definitions:

1. A *deterministic finite-state machine* is a system M=($\mathbf{X}$,$\mathbf{Y}$,$\mathbf{Q}$,f), where

   - $\mathbf{X}$ is a finite nonempty set of input symbols called the *input set* or *input alphabet.*
   - $\mathbf{Y}$ is a finite nonempty set of output symbols called the *output set* or *output alphabet.*
   - $\mathbf{Y}$ is a finite nonempty set of state symbols called the *state set* .
   - $f : \mathbf{X} \times \mathbf{Q} \rightarrow \mathbf{Q} \times \mathbf{Y}$ is a combined next-state and output function.

   Note. Traditionally, f, is divided into two functions, $f_q : \mathbf{X} \times \mathbf{Q} \rightarrow \mathbf{Q}$ and $f_y : \mathbf{X} \times \mathbf{Q} \rightarrow \mathbf{Y}$ called, respectively, the *next-state function* and the *output function.* For our purpose, it is more convenient to combine these two functions into a single function.

   The work of M is described in discrete time ν as follows :

   $$(q(\nu+1), y(\nu)) = f(x(\nu), q(\nu)); \tag{4}$$

   where, $y(\nu) \in \mathbf{Y}$, and $x(\nu), a \in \mathbf{X}$.

2. A *probabilistic finite-state machine* is a system M_p=($\mathbf{X}$,$\mathbf{Y}$,$\mathbf{Q}$,f_p), where

   - $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Q}$ are the same as before.
   - $f_p : \mathbf{X} \times \mathbf{Q} \times \mathbf{Q} \times \mathbf{Y} \rightarrow [0,1]$ is the combined function representing the next-state and output conditional probabilities. f_p satisfies the condition: for all $(x,q) \in \mathbf{X} \times \mathbf{Q}$ $\sum_{((q',y)\in \mathbf{Q}\times\mathbf{Y})} f(x,q,q',y) = 1$.

   The work of M_p is described in discrete time ν as follows :

   $$P(\ q(\nu+1) = c,\ y(\nu) = d \mid x(\nu) = a,\ q(\nu) = b\ ) = \ f_p(a,b,c,d); \tag{5}$$

   where, $x(\nu), a \in \mathbf{X}$, $q(\nu+1), q(\nu), c, b \in \mathbf{Q}$, and $y(\nu), d \in \mathbf{Y}$,

   Note. A deterministic finite-state machine is an extreme case of a probabilistic finite-state machine with $f_p(a,b,c,d) \in \{0,1\}$.

3. A behavior that can be represented in terms of a finite-state machine is called a *behavior of type 3.*

6.6 Behavior of type 3 as a behavior of type 4 with a feedback

It is easy to show that any behavior of type 3 can be represented as a behavior of type 4 with a one-step delayed feedback. Let M=(**X**,**Y**,**Q**,f1) be a finite-state machine, and let M1=(**X1**,**Y1**,f2) be a combinatorial machine, such that $\mathbf{X1} = \mathbf{X} \times \mathbf{Q}$ and $\mathbf{Y1} = \mathbf{Q} \times \mathbf{Y}$. The work of M1 is described as follows:

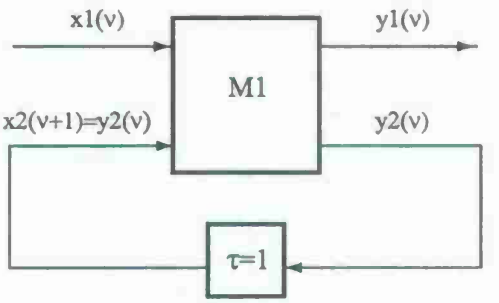

Figure 10: Replacing a finite-state machine by a combinatorial machine with a one-step delayed feedback.

$$(y1(\nu), y2(\nu)) = f1(x1(\nu), x2(\nu)); \tag{6}$$

where $x1(\nu) \in \mathbf{X}$, $x2(\nu) \in \mathbf{Q}$, $y1(\nu) \in \mathbf{Y}$, and $y2(\nu) \in \mathbf{Q}$.

Let us introduce a one-step delayed feedback from output $y2$ to input $x2$. We have

$$x2(\nu) = y2(\nu - 1); \tag{7}$$

Renaming $x1$ as x, $x2$ as q, $y1$ as y, $f1$ as f, and substituting (7) into (6) we get expression (4) describing the work of the finite-state machine M. Since function f1 can be selected arbitrarily, any deterministic finite-state machine can be implemented as a deterministic combinatorial machine with a delayed feedback. It is easy to see that the same holds for probabilistic machines.

7 Learning to simulate behavior of type 0 (Tasks 3 and 4)

7.1 The main steps for solving Task 3

We proceed in the following steps:

Step 1. Define the concept of a Turing machine and the notion of behavior of type 0.

Step 2. Show that the tape of a Turing machine can be replaced by a GRAM.

Step 3. Show that system of Figure 12 can be trained in an experiment of supervised learning to simulate the finite-state part of any Turing machine.

Step 4. Prove the theorem of Task 3.

7.2 The concept of a Turing machine: behavior of type 0

A *Turing machine* is a finite-state machine interacting with an external read/write memory – traditionally, a tape divided into squares (Figure 11). Theoretically, the tape has infinite length. However, as was explained in Section 1.2, this requirement is not important from the practical viewpoint. Even a finite tape cannot be treated as a finite-state machine because the number of states of such a tape explodes exponentially with the number of squares.

A classical Turing machine has a read/write head the position of which indicates the position of a single square of the tape called the *scanned square*. The address of the scanned square doesn't need to be defined explicitly as it is done in the case of a RAM, or a GRAM (Section 4.1). At each step the Turing machine can perform the following elementary operations:

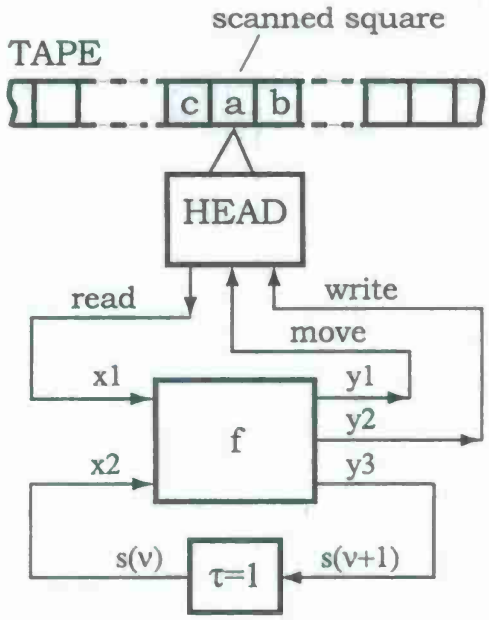

Figure 11: The general architecture of a Turing machine

1. Read a symbol from a scanned square.
2. Write a symbol into the scanned square.
3. Move the head one square to the right, one square to the left, or stay in the same square.
4. Change the current *state of mind* (see definition below) to the next state of mind.

Definition. A *finite-state part of a Turing machine* is a deterministic finite-state machine T=(**X**,**Y**,**S**,f,s0), where

X is the set of external symbols read from the tape;

S is the set of internal symbols representing the *states of mind* of a Turing machine. This set includes a special *halt* symbol H. A Turing machine stops when it enters the halt state.

$s0 \in \mathbf{S}$ is a symbol representing the initial state of mind of the Turing machine.

$Y = \mathbf{S} \times \mathbf{M} \times \mathbf{X}$ is the set of output symbols representing the reactions of the Turing machine. This set is the product of the three sets:

S is the set of the next states of mind (the same as the set of the current states of mind).

$\mathbf{M} = \{R, L, S\}$ represent the movements of the head, where R, L, and S mean, respectively, move one square to the right, move one square to the left, and stay in the same square.

X is the set of symbols that can be written on tape (the same as the set of symbols that can be read from the tape).

$f : \mathbf{X} \times \mathbf{S} \rightarrow \mathbf{Y}$ is the function represented as the set of quintuples called *productions* or *commands* of the Turing machine.

A behavior that can be represented in terms of a Turing machine will be called a *behavior of type 0*.

7.3 Learning to simulate the finite-state part of a Turing machine

It is convenient to redraw the lower part of Figure 4 including system AM, nuclei NM, and Teacher as shown in Figure 12. We also included the one-step delayed speech feedback $s(\nu) \rightarrow s(\nu - 1)$. Because of this feedback, system AM needs to work only as a combinatorial machine.

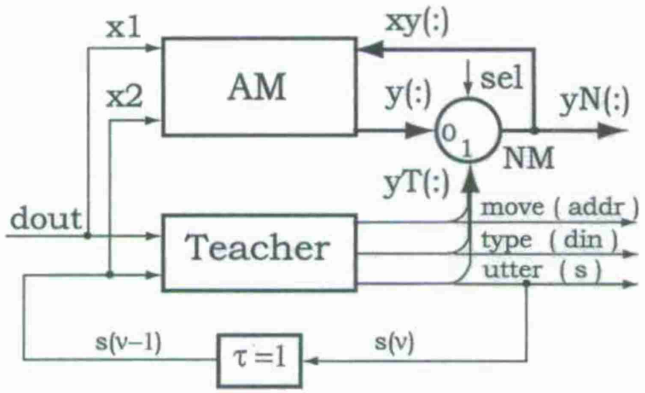

Figure 12: Training system AM of Figure 4 to execute the program of a Turing machine.

Remarks

1. It should be emphasized that the introduction of the speech feedback, $s(\nu) \rightarrow s(\nu - 1)$, allows us to treat the *states of mind*, $s(\nu)$, of the simulated Turing machine as *observable variables* available for learning. This makes it theoretically possible to learn to simulate the finite-state part of any Turing machine. The corresponding learning problem would be theoretically unsolvable if the states were treated as *hidden variables*.

2. In traditional theories of symbolic learning, the internal states of the teacher (the target machine) are treated as hidden variables the structure of which must be uncovered by a learning algorithm. We argue that this traditional approach is too restrictive when it comes to the problem of human learning. A human learner doesn't view a human teacher as an absolutely black box. The black box approach to learning leads to implications, such as, e.g., the Gold theorem [16], that are not supported by observations. As humans, we have a great deal of built-in information about the internal states of other humans. See Section 7.8.

To prove the theorem of Task 2 we will assume that system AM in Figure 12 is organized as Model 2.2 with $a = 0$, $m = 2$, and $p = 3$. As in Section 6.4, we divide the experiment into training and examination.

Training. Training starts at $\nu = \nu 0$ and lasts until $\nu = \nu 1 - 1$, where $\nu 1$ is the first moment of examination. The following conditions are satisfied:

1. $sel = 1$, and, accordingly, $xy(:) = yT(:)$.
2. At the beginning of training, LTM is empty. That is, for $i = 1 : n$ $gx(1,i)(\nu 0) = gx(2,i)(\nu 0) = \varepsilon$; $gy(1,i)(\nu 0) = gy(2,i)(\nu 0) = gy(3,i)(\nu 0) = \varepsilon$, and $wptr(\nu 0) = 1$.
3. Learning is enabled: $wen = 1$. We assume that the LTM is big enough to store all training sequence. That is, $n \geq \nu 1 - \nu 0$.

Examination. At this stage, $sel = 0$, and $xy(:) = y(:)$. For simplicity, we assume that AM does not learn during this stage, that is, $wen = 0$. This requirement is not necessary. If n is big enough, AM could continue recording data in its LTM.

Theorem 4. *System AM defined above learns to simulate the finite state part of any Turing machine, if each command of this machine is presented at least once at the stage of training.*

Proof. The theorem follows from the definition of a finite-state part of a Turing machine (Section 7.2), the Theorem 3 of Section 6.4, and the fact that any finite-state machine can be represented as a combinatorial machine with a one-step delayed feedback (Section 6.6). □

7.4 Replacing TAPE and HEAD by a modified GRAM

Let us compare the (TAPE,HEAD) system of the Turing machine shown in Figure 11 with the external system (W,D) shown in Figure 5. The *read* output is similar to *dout* output, the *write* input is similar to *din* input. However, the *move* input is different from *addr* input. To resolve this problem we need to slightly improve system (W,D) so it will be to correctly react to *move* symbols, $move \in \mathbf{M} = \{L, R, S\}$. The easiest way to achieve this goal is to slightly modify the formal description of GRAM from Section 4.1. Instead of expressions (4.1.1) and (4.1.2) we will use the following expressions:

(1) $if(move == R)\ addr = addr + 1;$
... $else\ if(move == L)\ addr = addr - 1;\ \ end$

(2) $if\ (din == \varepsilon)\ \ dout = mem(addr);$

(3) $else\ \ dout = din;\ mem(addr) = din;\ \ end$

Expression (1) can be interpreted as an addition of a transducer, TR, shown in Figure 13 to system (W,D) of Figure 5 a). The transducer works as a bidirectional counter that increments when $move = R$ and decrements when $move = L$. The input $move = S$ has no effect.

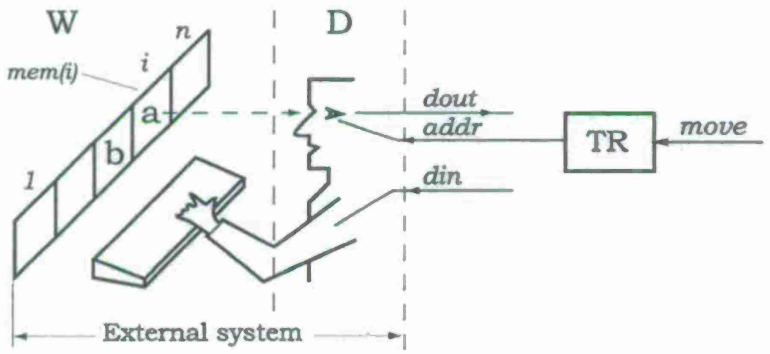

Figure 13: Adding transducer, TR, to make system (W,D) behave as (HEAD, TAPE) of a Turing machine in Figure 11.

7.5 Computations with the use of external tape

Consider the following experiment with the robot of Figure 3. Assume that the transducer TR shown in Figure 13 is introduced at the output of centers NM1. This output will produce the $move \in \mathbf{M} = \{L, R, S\}$ symbols of the simulated Turing machine.

Training. At this stage $NS.sel = 1$. The robot's eye is open and the motor system AM sees a symbol *dout* read from the tape (screen). The teacher forces the robot to perform several example of an algorithm with different input data presented on tape. The examples are selected in such a way that each command of simulated Turing machine is executed at least once.

Examination. At this stage, $NS.sel = 1$ – the eye is open. The teacher presents new data on tape, positions the robot's eye in the first square (traditionally, the leftmost square), and sets the robot's initial *state of mind* by forcing the robot to utter the initial symbol. From this moment on the teacher does not interfere and the robot performs computations automatically.

Theorem 5. *Let Model 2.2 be used as system AM of Figure 3. There exist parameters of AM such that after the described Training the robot of Figure 3 passes the described Examination.*

Proof. The result follows from the Theorem 4 of Section 7.3. □

7.6 Computations with the use of imaginary tape

In Section 5.2, we have shown that system AS of Figure 3, organized as Model 2.2, can learn to simulate any GRAM. The minimum length of the training sequence is $m \cdot n$, where m is the number of data symbols and n is the number of locations of the simulated GRAM – each data symbol needs to be written at least once in each location of the GRAM. For example it takes $2 \cdot 100 = 200$ steps to learn to simulate a tape with 2 symbols and 100 locations. This is a very short training sequence as compared to the number of states of the GRAM. The latter is $m^n = 2^{100}$ in the case of the GRAM simulating the above tape.

Using the described property of system AS we can enhance the experiments of training and examination described in the previous section to allow the robot to learn to perform computations without seeing the external tape. Let us modify the experiments of training and examination as follows. We assume that the robot will never use more than n locations of the external tape.

Training. This stage is similar to the training in the previous section except, the robot must be forced by the teacher to perform all *move* commands in each of the n locations of the tape. This means that the length of the training sequence increases roughly proportionally to n. (There is no combinatorial explosion of the type m^n.)

Examination. This stage is divided into two parts.

1. At this stage, $NS.sel = 1$ – the eye is open. The teacher presents new data on tape, positions the robot's eye in the first square.

2. The teacher forces the robot to scan new data, return to the first square, and utter the initial symbol. From this moment on $NS.sel = 0$ (the eye is closed), the teacher does not interfere, and the robot performs computations using the imaginary tape simulated by system AS.

Theorem 6. *Let Model 2.2 be used as both system AS and system AM of Figure 3. There exist parameters of AS and AM such that after the described Training the robot of Figure 3 passes the described Examination.*

Proof. The result follows from the Theorem 1 of Section 5.2 and the Theorem 4 of Section 7.3. □

7.7 The C++ program EROBOT (Task 4)

The C++ program called EROBOT allows the user to perform the experiments described in sections 7.5 and 7.6 with the robot of Figure 3. The simulations confirm that the corresponding theorems do hold.

The main screen of the program EROBOT performing the parentheses checker algorithm [20] is shown in Figure 14. The screen displays three dialog boxes representing the systems: (1) the external system (W,D) – the left box, (2) the mental imagery system AS – the upper right box, (3) the motor control system AM – the lower right box. The user can perform a broad range of experiments with the program and teach it to perform algorithms representable as Turing machines with up to 1000 commands.

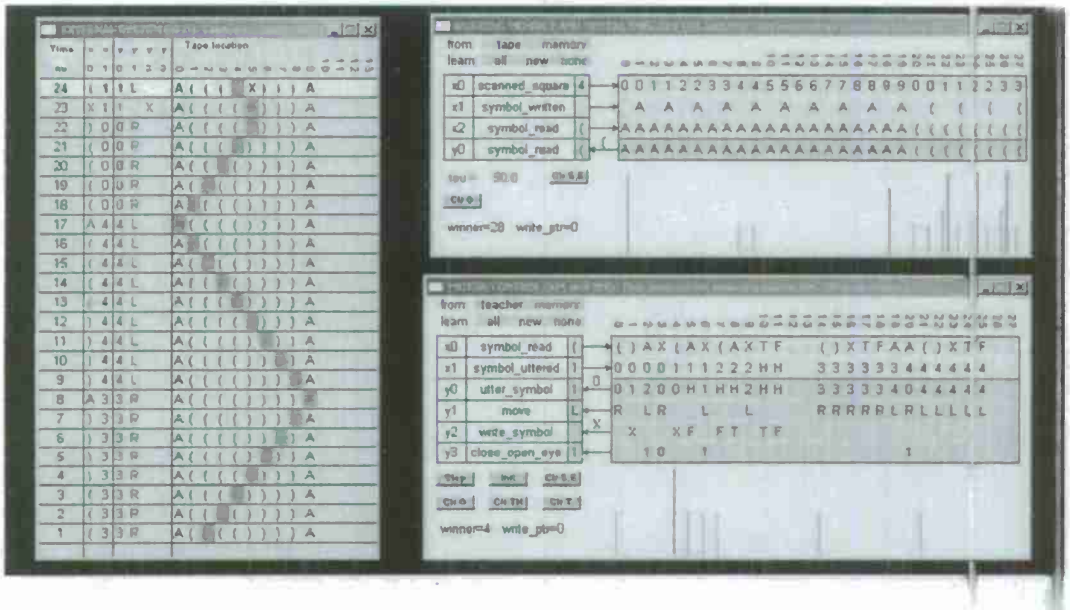

Figure 14: The main screen of the program EROBOT.

7.8 The myth that behavior of type 0 cannot be learned

As mentioned in Remark 2 in Section 7.3, some theories of learning equate the problem of learning with the problem of deciphering the structure of a target machine (teacher) observed as a black box by another machine (learner). Usually the target machine is treated as a grammar that has to be identified from the set of sentences [16, 15]. With this definition of learning, the learner cannot learn to simulate the behavior of the teacher of the type higher than type 3 (finite-state grammars).

This general result seems to contradict to our Theorem 1 of Section 5.2 showing that the behavior of a GRAM can be learned – a GRAM is a system of type 0. In fact, there is no contradiction. System AS in Figure 7 (learner) does not treat GRAM (teacher) as a black box. Defined as Model 2.2, system AS has some built-in information about the external system (W,D) – namely the effect of recency associated with the decay of $e(i)$ matches some fundamental physical properties of (W,D). Accordingly, the black box limitations do not apply. We argue that the same holds for the phenomenon of human learning. A human learner does not treat a human teacher, and other external systems as black boxes. It expects these systems to have certain properties. In fact, evolution created biological systems in such a way that they can survive in the real physical world.

Consider the sentence: *"I know how you feel"*. It would be impossible to teach people the meaning of such concepts as *"pain"*, *"pleasure"*, *"imagine"*, *"think"*, *"wait"*, etc., if our brain had absolutely no built-in knowledge about these concepts. A human teacher and a human learner have similar brains, so the learner knows a great deal about the internal states of the teacher. The whole notion of the *black-box-learning* is a fallacy.

IMPORTANT. If the behavior of GRAM can be learned, then, according to Theorem 6 of Section 7.6, any behavior of type 0 can be learned. The computer simulation confirms this criti-

cally important theoretical result. The result changes the whole attitude toward the problem of human symbolic learning. It means that to be able to learn anything (to be a learning system universal in Turing sense) the human neocortex must use a universal learning algorithm close, in a sense, to the complete memory algorithm used in Model 2.2. Trying to invent a "supersmart" learning algorithm leads to principal limitations on what can be learned. We argue that the principal limitations of the traditional symbolic learning systems stem from this pitfall.

8 On a neural implementation of primitive E-machines (Tasks 6-9)

8.1 An example of an associative neural network with temporal modulation

It is helpful to have an intuitive link between E-machines and neural networks. Such a link provides a source of neurobiological heuristic considerations for the design of the models of E-machines and the source of psychological heuristic considerations for the design of the corresponding class of neural models. Figure 15 shows the general architecture of a homogeneous neural network corresponding to a PEM. The architecture was discussed in [7, 9].

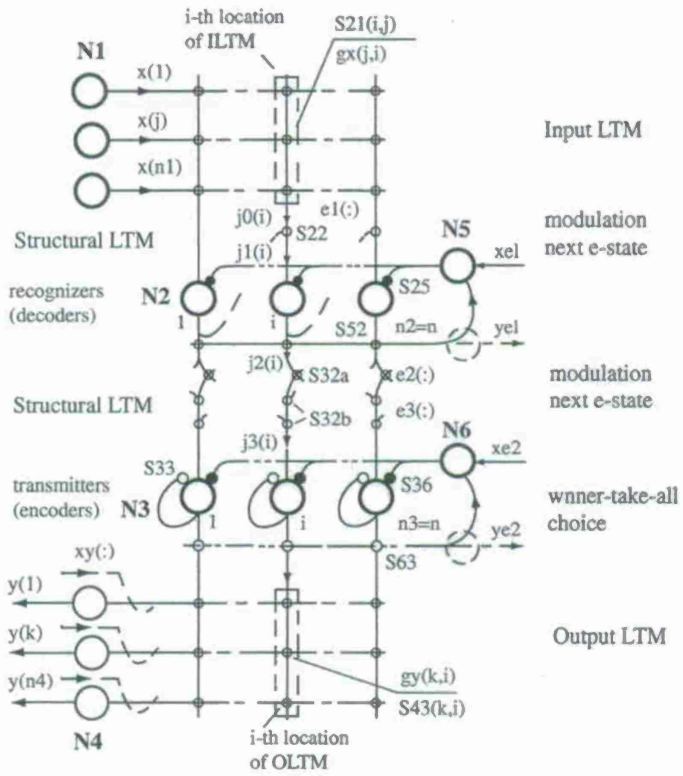

Figure 15: An example of a neural implementation of a primitive E-machine

Large circles with incoming and outgoing lines represent *centers* – elements that are assigned certain coordinates in the network. A center with incoming and outgoing lines can be interpreted

as a neuron with its dendrites and axons, respectively, or a network functionally equivalent to a "large neuron". Small circles represent *couplings* – the elements whose position in the network is described by a pair of centers communicating through this coupling. A coupling can be interpreted as a synapse or a circuit functionally equivalent to a "large synapse". The white and the black small circles represent excitatory and inhibitory synapses, respectively. In what follows we use the terms neuron and synapse instead of the terms center and coupling, respectively [4].

Figure 15 displays six sets of neurons (N1, N2, N3, N4, N5, and N6) and 10 sets of synapses (S21, S22, S25, S52, S32a, S32b, S33, S36, S63, S43).

Nj(i) is the i-th neuron from the j-th set. Skja(n,m) is the synapse between neurons Nj(m) and Nk(n), where 'a' is an additional index describing the type of synapse (in case there are several different types of synapses between the sets of neurons Nj and Nk). The ':' substituted for an index indicates the whole subset of elements (variables) corresponding to the entire set of values of this index. S21(i,:)≐(S21(i,1),...S21(i,n1)). S21(:,:) is the same as S21, etc.

The diagram depicts three types of synapses with E-states: $e1(:)$,$e2(:)$, and $e3(:)$, serving different purposes. $e2(:)$ is similar to the $e(:)$ of Model 2.2; e1(:) describes the effect of lateral modulation that allows the network to decode temporal sequences; e3(:) describes spreading residual excitation producing the effect of broad temporal context. **Remark.** This neural network corresponds to a PEM more complex than Model (2.2) [7, 9].

The Input LTM and the Output LTM are implemented, respectively, as the input, and output synaptic matrices, $S21(:,:)$ and $S4(:,:)$. The network also has some intermediate synaptic memory called Structural LTM (SLTM). This memory corresponds to modifiable connections among neuron decoders, N2, and neuron encoders, N3.

Layer N3 implements a *winner-take-all* (WTA) choice. The layer has local excitatory feedbacks, S33, and a global inhibitory feedback via neuron N6 – the, so-called, *inhibit-everyone-and-excite-itself* principle. The WTA choice could also be implemented by using inhibitory synapses S33(i2,i1) with gains ≥ 1.0 for all synapses except those with $i2 = i1$ – the, so-called, *inhibit-everyone-but-itself* principle. In the next section we explicitly describe the dynamics of the WTA. In the discussed case analytical solution is possible.

8.2 The dynamics of the winner-take-all choice

To explore the dynamics of the winner-take-all (WTA) layer let us consider a simplified version of the network of Figure 15 shown in Figure 16. Unlike the network of Figure 15 the simplified model does not have slow temporal modulation (the E-states). The differential equations describe the the fast dynamics of the postsynaptic potential of neurons N2. All other elements are described by algebraic equations. Neurons N2 are treated as first-order linear threshold elements with the threshold equal to zero. To produce a step-wise (E-machine-like) performance the model has a periodic inhibitory input x_{inh}. The period of this input, call it T_{inh}, must be much larger than the time constant of N2, τ ($T_{inh} \gg \tau$) to allow the transient process to reach its stable state during each step. Each neuron of layer N2 has an excitatory feedback with the gain α, and an inhibitory feedback with the gain β. g^x_{ij} and g^y_{ki} denote the gains of synapses S21(i,j) and S32(k,i), respectively. d_i is the output of neuron N2(i).

[4] We use the term *coupling* rather than the traditional term *connection* to emphasize that, in general, we treat a synapse as a complex nonlinear dynamical element with E-states, rather than just a connection with a variable weight.

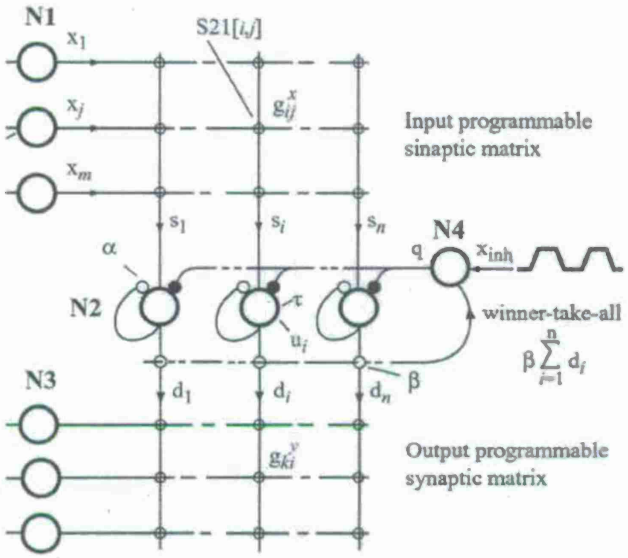

Figure 16: A simplified version of the network of Figure 15 illustrating the WTA dynamics.

$$s_i = \sum_{j=1}^{m} g_{ij}^x \cdot x_j \tag{1}$$

$$\tau \frac{du_i}{dt} + u_i = s_i + \alpha \cdot d_i - q \tag{2}$$

$$d_i = \begin{cases} u_i & \text{if } u_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$q = \beta \sum_{i=1}^{n} d_i + x_{inh} \tag{4}$$

$$y_k = \sum_{i=1}^{n} g_{ki}^y \cdot d_i \tag{5}$$

Let all x_j and x_{inh} (and, therefore, all s_i) be step functions of time. Then, for all active neurons from layer N2 (the neurons for which $u_i > 0$), the solution of equations (2)–(4) can be represented in the following explicit form:

$$\begin{aligned} u_i = &\frac{(s_i - s_{av})}{\alpha - 1}(e^{at} - 1) + (u_i^0 - u_{av}^0)e^{at} \\ &+ \frac{(s_{av} - x_{inh})}{1 + \beta \cdot n_1 - \alpha}(1 - e^{-bt}) + u_{av}^0 \cdot e^{-bt} \end{aligned} \tag{6}$$

where

- n_1 is the number of active neurones from N2.
- u_i^0 $(i = 1, ...n)$ are the values of u_i at t=0.

- s_{av} and u^0_{av} are the average values of s_i and u^0_i for all active neurons from N2. That is,

$$s_{av} = \frac{1}{n_1}\sum_{i=1}^{n_1} s_i \tag{7}$$

$$u^0_{av} = \frac{1}{n_1}\sum_{i=1}^{n_1} u^0_i \tag{8}$$

Parameters a and b in e^{at} and e^{-bt} are as follows:

$$a = (\alpha - 1)/\tau \tag{9}$$

$$b = (1 + \beta \cdot n_1 - \alpha)/\tau \tag{10}$$

Let $1 < \alpha < 1 + \beta$. Then $a > 0$ and $b > 0$. According to expression (9), neurons $N2[i]$ with $s_i > s_{av}$ increase their potentials u_i. Neuron $N2[i]$ with $s_i < s_{av}$ decrease their potentials and switch off once $u_i < 0$. This reduces n_1 and increases s_{av} making $s_i < s_{av}$ for some additional neurons from N2. Eventually, only neurons with $s_i = max(s_1, ...s_n)$ will have $u_i > 0$. it can be shown that this equilibrium is unstable if $n_i > 1$. Therefore, in the presence of noise, at the end of the transient response there will be only one winner randomly selected from the set of neurons with the maximum level of s_i.

8.3 The C++ program WTA

The WTA program simulates the dynamics of a winner-take-all layer described by equations 2, 3, and 4. The main screen of the program is shown in Figure 17. The upper dialog box displays the periodic inhibitory input, x_{inh}. The lower dialog box displays the similarity front, s (yellow lines), the positive postsynaptic potential, u, of competing neurons (magenta lines), and the negative postsynaptic potential of the neurons that are turned off by the inhibitory feedback (red lines). If both α and β are bigger than one, only one neuron with the maximum s_i wins the competition.

Remark. We are currently working on the extension of the WTA program that would allow the user to explore the competition of spiking neurons. As is known, the winner-take-all choice can be efficiently implemented in a layer of spiking neurons. Besides being biologically more consistent, the latter approach has a number of information processing advantages for the neural implementation of E-machine.

8.4 What is a neurobiological interpretation of E-states?

Theoretically, any dynamic variable with a time constant significantly larger than a "psychological" time step, $\Delta t \approx 10 - 100msec$, can provide a physical implementation of a certain type of phenomenological E-state. The psychological time step can be contrasted with a "neurobiological" time step, $dt \to 0$. For the purpose of a real time computer simulation, dt can be on the order of $dt = 1 - 10\mu sec$. We believe that cognitive-level models should be made as independent of the fast neural dynamics as possible.

The majority of the differential equations used in traditional ANN models can be interpreted as describing either the accumulation of chemicals in cellular compartments or the accumulation of

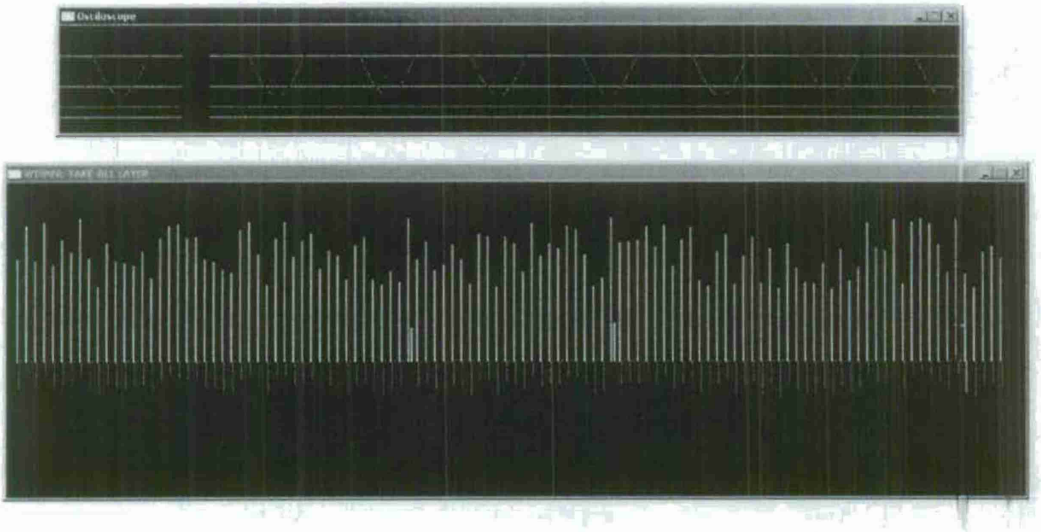

Figure 17: The main screen of the WTA program.

charges on the surfaces of cellular membranes. The latter dynamics have a small time constant (in the microsecond range) and cannot be used to implement E-states. In contrast, the accumulation of chemicals may have much bigger time constants. We argue however, that the slow dynamics of chemical concentrations per se does not provide the mathematical sophistication needed for implementing nontrivial next E-state procedures.

In what follows we discuss a possibility of connecting the dynamics of phenomenological E-states with the statistical conformational dynamics of ensembles of membrane proteins. The approach represents a natural system-theoretical extension of the classical Hodgkin and Huxley theory (HH theory) [14].

In this approach, a single protein molecule is treated as a microscopic probabilistic machine (a Markov system) with transitional probability densities depending on various macroscopic variables such as membrane potential and concentrations of different chemicals (neurotransmitters). The microscopic outputs of all molecules in a certain conformation (state) are summed up to produce the total macroscopic output of an ensemble of microscopic machines in this state. In the case of an ensemble of ion channels, the output is the total ion current. In the case of an ensemble of enzymes, the output is the flow of a chemical (e.g., the second messenger).

The time constants of the macroscopic dynamics of the ensembles of the above machines depend on the values of the transitional probability densities. The smaller the densities the bigger the time constants. In the case of the HH model the time constants for sodium and potassium channels are rather small (milliseconds). In more complex cases, however, it is not unreasonable to postulate time constants ranging from split seconds, to minutes, to hours, and even to days [13].

8.5 The statistical molecular dynamics of E-states

To formalize the above intuitive description we need to introduce the following definitions.

Definition. A *Probabilistic Molecular Machine* (PMM)[5] is a system $(\mathbf{X}, \mathbf{Y}, \mathbf{S}, \alpha, \omega)$, where

- $\mathbf{X}$ and $\mathbf{Y}$ are the sets of real input and output vectors, respectively
- $\mathbf{S}$= $\{s_0, \ldots, s_{n-1}\}$ is a finite set of states
- $\alpha : \mathbf{X} \times \mathbf{S} \times \mathbf{S} \rightarrow \mathbf{R}'$ is a function describing the input-dependent conditional probability densities of state transitions, where
  $\alpha(x, s_i, s_j)dt$ is the conditional probability of transfer from state s_j to state s_i during time interval dt,
  $x \in \mathbf{X}$ is the value of input, and $\mathbf{R}'$ is the set of non-negative real numbers. The components of x are called *generalized potentials*. They can be interpreted as membrane potential and/or concentrations of different neurotransmitters.
- $\omega : \mathbf{X} \times \mathbf{S} \rightarrow \mathbf{Y}$ is a function describing output. The components of y are called *generalized currents*. They can be interpreted as ion currents and/or the flows of second messengers.

Let $x \in \mathbf{X}$, $y \in \mathbf{Y}$, $s \in \mathbf{S}$ be, respectively, the values of input, output, and state at time t, and let P_i be the probability that $s = s_i$. The work of a PMM is described as follows:

$$\frac{dP_i}{dt} = \sum_{j \neq i} \alpha(x, s_i, s_j) P_j - P_i \sum_{j \neq i} \alpha(x, s_j, s_i) \tag{11}$$

$$at\ t = 0 \qquad \sum_{i=0}^{n-1} P_i = 1 \tag{12}$$

$$y = \omega(x, s) \tag{13}$$

Summing the left an the right parts of equations 11 for $i = 0, \ldots, n-1$, it is easy to verify that condition 12 holds for any t.

Definition. An *Ensemble of Probabilistic Molecular Machines* (EPMM) is a set of identical independent PMMs with the same input vector, and the output vector equal to the sum of output vectors of individual PMMs.

Let N be the total number of PMMs, N_i be the number of PMMs in state s_i (the occupation number of state s_i), and let $e_i = N_i/N$ be the relative occupation number of state s_i.
We have

$$y = N \sum_{i=0}^{n-1} e_i \omega(x, s_i) \tag{14}$$

The behavior of the average $\overline{e}_i$ is described by the equations similar to (11) and (12).

$$\frac{d\overline{e}_i}{dt} = \sum_{j \neq i} a_{ij}(x) \overline{e}_j - \overline{e}_i \sum_{j \neq i} a_{ji}(x) \tag{15}$$

$$at\ t = 0 \qquad \sum_{i=0}^{n-1} \overline{e}_i = 1 \tag{16}$$

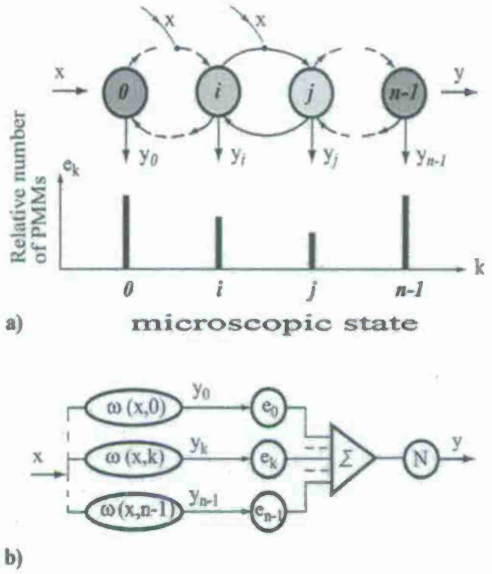

Figure 18: The interpretation of the macroscopic E-states as the occupation numbers of the microscopic states of a PMM.

where $a_{ij}(x) = \alpha(x, s_i, s_j)$ describes the rate constant of transfer between states s_j and s_i.

The structure of equations 14 and 15 is illustrated in Figures 18 and 19. The diagram of Figure 19 suggests that an EPMM provides a statistical implementation of a mixed-signal computer with the rate constants serving as input-controlled coefficients (that can serve as switches).

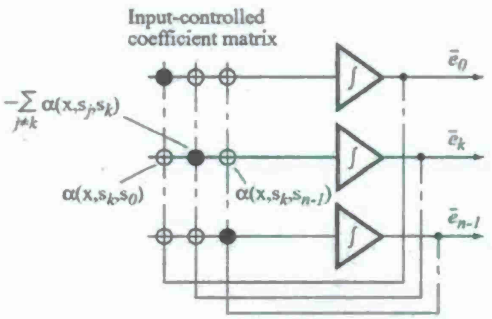

Figure 19: An EPMM as a mixed signal statistical computer.

This implementation is extremely robust because the properties of the whole computer are determined by the properties of a single PMM. No interaction among PMMs is needed. Variable external connections are replaced by variable internal probabilities, and statistics does the trick. It is tempting to say that, in the same way as statistical mechanics of simple molecules leads to thermodynamics, the statistical mechanics of very complex molecule-machines leads to neural computations.

[5]One can also interpret the abbreviation PMM as standing for *Protein Molecule Machine.*

Another obvious advantage of the EPMM implementation of the dynamics of E-states is that this implementation relies on molecular time – the time constants do not depend on volatile parameters of neurons, synapses, and neural networks.

8.6 The C++ program EPMM

The EPMM program allows the user to simulate the dynamics of a cell with up to 10 different types of PMM each having up to 18 states. Two modes of simulation are supported: (1) the differential equation mode ($n \to \infty$), and (2) the Monte Carlo mode with up to n=10000.

Figure 20 presents the main screen of the program in the course of simulation of the spike dynamics in an HH-type of neuron. The sodium and potassium channels are represented as PMM's with 5 states shown on the screen. The user can change the structure of the PMM's on the fly and immediately see how this change affects the shape and the frequency of spikes.

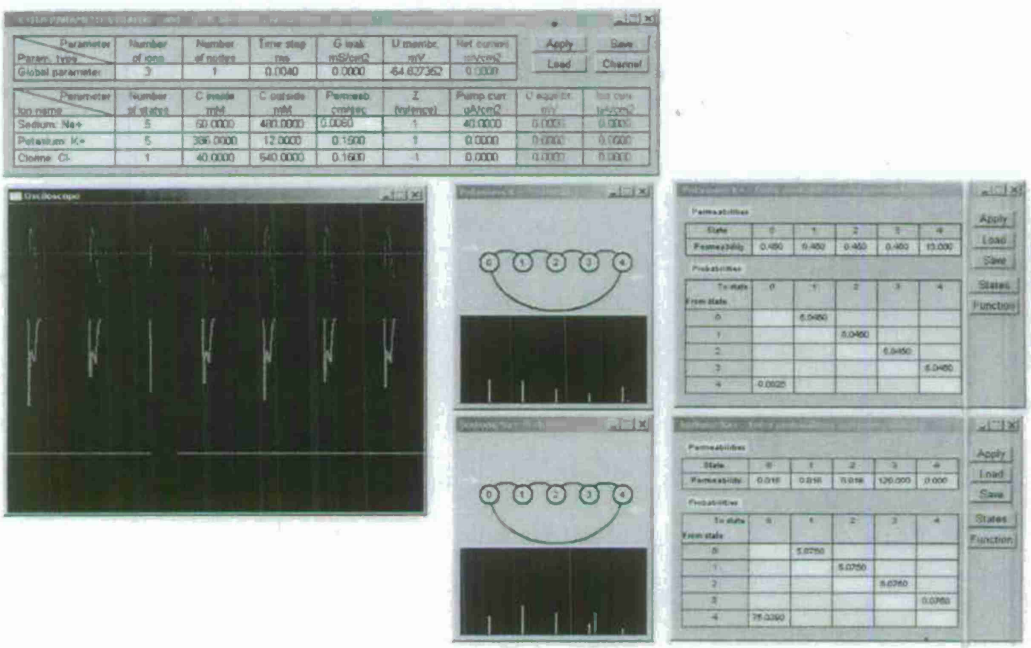

Figure 20: The main screen of the EPPM program.

Remarks:

1. The EPMM formalism gives one a tool for implementing a broad range of sophisticated next E-state procedures. The complexity of such procedures depends on the postulated structure of the PMM's and on the way the EPMM's interact in a cell via membrane potential and second messengers.

2. We use the HH theory as a test case for the EPMM formalism. In the example shown in Figure 20, two simple PMMs corresponding to sodium and potassium channels, interacting via common membrane potential, produce quite sophisticated spike dynamics. Much more complex dynamics can be implemented in a similar way by using more complex PMMs.

3. Using the software similar to EPMM and EROBOT, a broad range of neurobiological and psychological implications of different hypotheses about the properties of membrane proteins can be explored via computer simulation. Accordingly, the discussed approach offers a solid foundation for a multidisciplinary research aimed at the development of a biologically consistent and predictive (rather than merely biologically inspired) mathematical theory of the brain.

9 Promising directions of research

There are many interesting problems and possibilities associated with the development, exploration, and understanding the symbolic/dynamical computational paradigm represented by E-machines. These include:

1. **Decoding temporal sequences.** Adding lateral pre-tuning to the next E-state procedure addresses this problem. The corresponding PEM can learn to simulate, in principle, any output independent finite memory machine. Introducing a delayed feedback in the above PEM leads to a system capable of learning to simulate any output dependent finite memory machine [7, 9].

2. **Waiting associations.** One can add an E-state creating an effect of waiting association. In this way one can produce an effect of a finite stack without using a RAM buffer, and simulate context-free grammars with limited memory. This allows one to get an effect of calling and returning from subroutines [7].

3. **Broad temporal context.** This effect can be produced by adding different types of "spreading" E-states. Changing the time constants and the radii of spread of such E-states leads to a broad range of effects of context-dependent mental set [7, 9, 33].

4. **Active scanning of associative memory.** People can answer the questions about what happened *after* or *before* a certain event. This effect can be achieved by first creating an E-state profile that activates the information about the mentioned event, and then actively shifting this profile in the time-wise or counter-time-wise direction. This can be done by introducing additional control inputs in a PEM.

5. **Inhibiting data with given features.** One does not have to think about the E-states as "excitations" or "activations". One can imagine the E-states producing inhibiting, pre-inhibiting, pre-activating, post-activating, etc. effects. In fact, any functions over sets of data stored in the LTM that assign some dynamic labels to the subsets of data can be thought of as different kinds of E-states. For example, one can imagine a situation in which one first creates an E-state activating a set of data (by addressing this data by content), and then temporarily inhibits the selected set of data by sending some inhibiting control input. The brain has many different neurotransmitters and receptors that may justify different hypotheses about the possible types of such control inputs. This again points to the importance of complex molecular computations mentioned in Section 8.6.

6. **From signals to symbols and vice versa.** The most likely candidate for the E-machine paradigm is the neocortex. In this metaphor, system AM corresponds to the frontal lobe, whereas system AS corresponds to the other three lobes as shown in Figure 21.

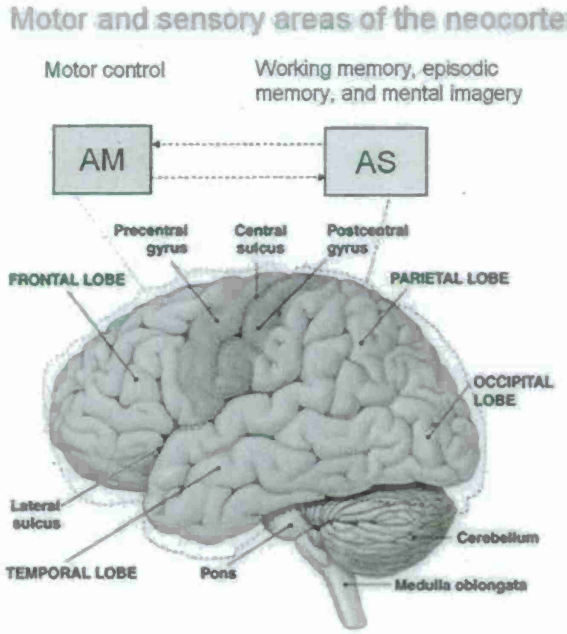

Figure 21: The $AM \Leftrightarrow AS$ architecture of the human neocortex

To make the E-machine paradigm practically implementable at the level of the neocortex, the lower levels of the brain must be able to convert sensory signals into sensory symbols, and motor symbols into motor signals. One of the possibilities is that the *signal-to-symbol* conversion is done by different feature detectors. Each detector would have a fixed sparse ID serving as a unique pointer to this detector. The sets of such IDs would form primary sensory alphabets. Similarly, the units generating primary motor features would have sparse IDs forming primary motor alphabets. With this approach, the higher association areas of the neocortex could deal mostly with symbols represented by the sparse IDs.

6. **High dimensionality of sensory devices. On the blessing of dimensionality.** One of the striking differences between the signal processing in the biological brain and that in artificial cognitive systems is the dimensionality of inputs and outputs. As an example it is helpful to consider a typical automatic speech recognition (ASR) system and compare it with the human speech recognition (HSR) system [30]..

A typical ASR system described in [31] starts with the 256-dimensional input vector representing short-term Fourier coefficients of the overlapping speech frames sampled 100 times per second. After several steps of intermediate processing, this 256-dimensional vector is transformed into, 39-dimensional vector consisting of 12, so-called, cepstral coefficients (and their two time derivatives), and the, so-called, log frame energy (and its two time derivatives). This 39-dimensional vector is presented to the higher levels of the ASR system for

further processing.

The HSR system starts with $\approx$ 4000-dimensional vector representing the output of the inner hair cells in cochlea of each ear. See Figure 22. This vector can be thought of as some counterpart of the 256-dimensional vector of the short-term Fourier coefficients of the ASR system. The number of cells processing the above auditory input expands to $\approx 90,000$ cells in the cochlear nucleus, to $\approx 390,000$ cells in the inferior colliculus, to $\approx 580,000$ cell in the medial geniculate body of the thalamus, to $\approx 100,000,000$ cells in the primary auditory cortex. The firing of the subsets of the latter cells represent the auditory world to the higher levels of the neocortex. *How can the higher levels make sense of this extremely high-dimensional input?*
It is intuitively obvious that there must exist some *blessing of dimensionality* that is efficiently utilized by the higher levels of the HSR system. The metaphor "neocortex as an E-machine" provides an insight into the nature of this blessing of dimensionality. The neocortex is a massively parallel system. It does not have time to compute different auditory features on the fly. However, it can efficiently pre-tune the subsets of the already formed feature detectors by changing the E-states. The more feature detectors are formed in the primary auditory cortex, the easier it is for the higher levels of the neocortex (organized as E-machines) to pre-tune the needed subsets of feature detectors depending on context.

Intuitively, the combinatorial possibilities of parallel dynamic reconfiguration associated with the "nonclassical" symbol processing paradigm employed by E-machines (dealing with functions representing subsets of pointers to activate subsets of "immovable symbols") overcomes the *curse of dimensionality.* In contrast, this curse plagues all systems that rely on 'classical" symbol manipulation techniques (dealing with pointers to move symbols in memory). This holds for the higher levels of processing in all existing ASR systems we are aware of.

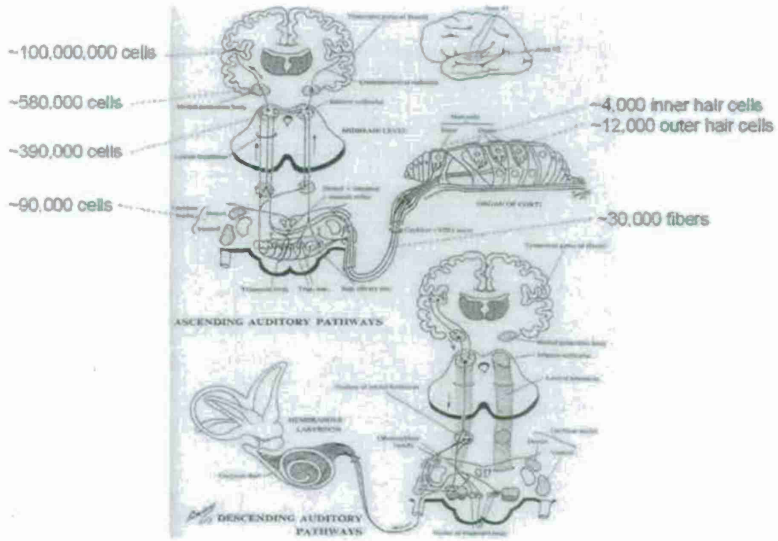

Figure 22: Human auditory pathways: there must exist some blessing of dimensionality.

7. **Sparse recoding in a hierarchical structure of associative memory**. As mentioned before, a complex E-machine (CEM) is a hierarchical associative learning system built from several PEMs. The concept of sparse IDs can be extended to allow the PEMs of higher levels to store data in terms of sparse-recoded references to the data stored in the PEMs of lower levels. This would naturally produce different effects of data compression and chunking (see, for example, [1]).

8. **Communication via association fibers**. The sparse encoding of symbols addresses the question of how the homogeneous areas of the neocortex of different modalities and levels (treated as PEMs) could efficiently communicate via association fibers. The numbers of association fibers are not big enough to provide crossbar connectivity. In the case of sparse encoding of symbols, several symbolic messages could be sent simultaneously with a low level of crosstalk. This would allow any small subset of talkers to broadcast their sparse-encoded messages simultaneously to large numbers of listeners. The only limitation is that not too many talkers must talk at the same time through the same set of association fibers. As an example, our estimate shows that around $m \approx 50$ neuron-talkers can talk simultaneously to $n \approx 10^9$ neuron-listeners through $k \approx 10^4$ association fibers. That is, *any-n1-to-any-n2* communication is physically impossible if $n1 = n2 = 10^9$ – it would require 10^{18} connections. However, *any-m-of-n1-to-any-n2* communication is quite possible if $m \approx 50$, and $n1 = n2 \approx 10^9$.

9. **Computing statistics on the fly depending on context**. At the symbolic level, the brain may not pre-compute statistics at the time of learning because statistics depends on context. *How could the neocortex compute statistics on the fly depending on context?* A sparse encoding of symbols offers a solution to this problem. If we change the procedures of CHOICE and ENCODING to allow several sparse-encoded symbols to be read simultaneously from the locations of OLTM with a "high enough" level of activation, summing up several sparse vectors produces a statistical filtering effect [7].

10. **Natural language**. This is the most challenging and interesting problem that, we believe, is well suited for the E-machine paradigm. It takes less information to dynamically activate the data structures that are already present in the LTM than to create new data structures. Even less information is needed if some data structures are already pre-tuned through the inputs of other modalities. Accordingly, unlike the statements of a formal language, the sentences of a natural language do not need to carry complete information. A hint can be sufficient to remove ambiguity in a given context. This sheds light on why people with similar backgrounds (G-states) and mental sets (E-states) can efficiently communicate via short messages, whereas people with different backgrounds and mental sets have difficulties understanding each other.

11. **Emotions and motivation**. We should mention the problem of emotions and motivation. *Can the E-machine paradigm shed light on this problem?* Let us postulate that, at a higher level, there exists some symbolic representation of emotions – otherwise our language would not have the names for our emotional states. If this postulate is correct, the E-machine formalism can be applied to the higher level learning involving emotions. People remember their pleasant and unpleasant emotional states. This means that, at a higher level, the effect

of positive/negative reinforcement cannot be reduced to the effect of increasing/decreasing the weights of sensorimotor associations. We postulate that, at a higher level, the brain forms associations involving the symbols representing emotions and other observable *internal* (I) states. The sets of these associations can be dynamically reconfigured depending on context by changing the E-states. This helps to understand why our concepts of "good" and "bad" depend on our knowledge and mental set.

It is easy to imagine a situation when retrieving emotional symbols affects control inputs that change the E-states that, in turn, affect the retrieval of emotional symbols, and so on. This would shed light on the nature of various self-reinforcing loops, such as the well known panic attack loop.

12. **Long-term learning. The utility problem**. One of the most difficult and, practically, most important problems encountered by traditional AI-type theories of symbolic learning is the, so-called, *utility problem* [29, 27, 28] – after a sufficiently long time of learning, the performance of a learning system begins to deteriorate. In contrast, human performance dramatically improves with learning. Our ability to learn increases with learning. We learn how to learn and even how to learn how to learn. The more knowledge we acquire the easier it becomes to learn more. Intuitively, this nonlinear effect is the result of learning with the references to what has already been learned. It also is a result of the brain's remarkable ability to use knowledge acquired in one context in a large number of other contexts – the, so-called, transfer of experience between contexts.

    We believe that the E-machine type approach to the problem of symbolic learning and context – learning is accumulation of symbolic knowledge in LTM and context is a dynamic reconfiguration of this knowledge by the functions represented by E-states – is well suited for tackling the problem of symbolic long-term learning. We argue that only a universal learning algorithm – an algorithm close, in a sense, to a complete memory algorithm used in Model 2.2 – can provide the expanding ability to learn and avoid the utility problem.

    Remarks:

    1. It is important not to confuse the *behavior aimed at learning* (e.g., studying at the Stanford university) with a *learning algorithm = data storage algorithm*. In a Turing universal learning system, a simple universal learning algorithm can, theoretically, produce arbitrarily complex behavior aimed at learning. We argue that the critical issue for the human-like long-term learning is what information to learn rather than how to store this information in the learner's LTM. Somehow, this critical issue is largely ignored when machine learning is compared with human learning. There is a "gold rush" for the development of all kinds of smart learning algorithms – hundreds of different learning algorithms are already invented. The general believe motivating this research is that the power of the brain is a result of some special learning algorithm(s).
    2. It is easy to see that the vision of a "supersmart" learning algorithm is a fallacy. Doing too much pre-processing of "raw" learning information before storing this information in LTM is not a good strategy in the case of a system that must deal with a very large number of contexts. No data storage algorithm can, in principle, know in advance what information may become useful in future contexts. Therefore, a system capable

of efficiently dealing with a very large number of different contexts must rely on the power of an efficient dynamic interpretation (decision making) procedure rather than on a "smart" data storage algorithm. The catch is that the loss of information at the time of learning is irremediable at the time of decision making. Theoretically, a powerful enough interpretation procedure can always make up for a dumb but universal learning algorithm – such as a complete memory algorithm. In contrast, no interpretation procedure can make up for a smart learning algorithm that loses information.

10 List of topics for Phase II (Task 10)

Here is an extended list of possible topics for the Phase II of the MoB project. The specific topics will be selected depending on the available resources.

1. **Setting real-time simulation environment**

   The C++ simulation environment on a Windows XP personal computer was sufficient for proving concepts at Phase I. In fact, we were able to provide a mathematical proof of the main concepts. This combination of theoretical studies and small scale computer simulations is not sufficient for Phase II.

   In most cases, the behavior of large E-machines cannot be understood theoretically. To make a serious progress toward understanding the cognitive possibilities of this class of systems we need to set a development environment allowing real time simulation of hierarchical E-machines with the size of LTM on the order of 10-100GB. Our estimates of the size of the human brain's LTM are on the order of 10-100TB.

   We would like to be able to do several large scale simulations demonstrating the efficiency of the E-machine paradigm for a broad range of cognitive problems that are difficult to solve using conventional computers[6]. These include:

2. **Motor Control**

   (a) **Afferent synthesis of sequential movements.**

   Synthesizing a complex sequential motor program by pre-tuning the elements of the already learned motor programs. **Example**: babbling syllables and then learning to produce words consisting of these syllables.

   (b) **Afferent synthesis of parallel movements.**

   Synthesizing a complex parallel motor program by sequentially pre-tuning parallel components of this program and then executing the whole program in parallel. **Example**: learning to play piano by training different hands and then producing two hand movement in parallel.

[6]What is critically important is that (with an appropriate hardware that can be developed at Phase III), the E-machine paradigm is scalable to the size of human neocortex. To our best knowledge, this is not true in the case of the majority of traditional ANN and AI algorithms.

(c) **Inverse kinematics**. Imitating a multi-link arm movement by being shown only the trajectory of the end of the last link. The trajectory can be executed in many different ways. The effect is achieved by pre-tuning the elements of the already learned trajectories. No trigonometric computations used in the traditional approaches to reverse kinematics are allowed – the brain cannot perform such computations.

(d) **Controlling parameters of movements by speech instructions.**

(1) Single word instructions: *faster*, *slower*, *stop*, *left*, *right*, etc.

(2) Simple sentences. *Wait for x1. Find x2.* Move to x3. Etc. The system must learn to execute these instructions, e.g., *Wait for* .., with the objects *z1,z2,..* which were never presented in the context of these instructions.

3. **Active context-dependent pattern recognition.**

   The system must learn to perform the actions that lead to pattern recognition in a given context. This is a more sophisticated approach to pattern recognition than just classification. The system must be able to learn to execute instruction: *What is it?*

4. **Mental set**.

   The system must interpret the same picture in different ways depending on its mental set. Example: the Necker cube.

5. **Language controlled mental imagery**.

   The system must be able to generate different mental images depending on speech instructions.

6. **Effects of actively scanning LTM**.

   The system must learn to scan LTM in response to questions: what was *before* event x, what was *after* event x, etc.

7. **Communication among primitive associative memories in a hierarchical associative memory**.

   Demonstrating effects of data compression and context-dependent statistical filtering resulting from the recoding of associations of lower levels.

8. **Comparing the amounts of computations for cognitive-level and neural-level simulations.**

   The metaphor "neocortex as an E-machine" suggests that not all details of neural implementation are important at the cognitive level. It is interesting to compare the amounts of computations needed for cognitive-level and neural-level simulations. Our preliminary estimates show that, at the cognitive level, the amount of computations can be reduced by the factor of 100-1000.

References

[1] Anderson, J.R. (1976). Language, Memory, and Thought. Hillsdale, New Jersey: *Lawrence Erlbaum Associates, Publishers.*

[2] Asimov, I. (1950) I, Robot. *Ballantine Books, New York.*

[3] Baddeley, A.D. (1982). Your memory: A user's guide. *MacMillan Publishing Co., Inc.*

[4] Burns, B.D. (1958). The Mammalian Cerebral Cortex *Arnold Publishers.*

[5] Chomsky, N. (1956). Three models for the description of language. *I.R.E. Transactions on Information Theory.* JT-2, 113-124.

[6] Eliashberg, V. 1967. On a class of learning machines. *Moscow: Proceedings of VNIIB, #54,* 350-398.

[7] Eliashberg, V. (1979). The concept of E-machine and the problem of context-dependent behavior. *TXU 40-320, US Copyright Office. Available from www.brain0.com.*

[8] Eliashberg, V. (1981). The concept of E-machine: On brain hardware and the algorithms of thinking. *Proceedings of the Third Annual Meeting of Cognitive Science Soc.*, 289-291.

[9] Eliashberg, V. (1989). Context-sensitive associative memory: "Residual excitation" in neural networks as the mechanism of STM and mental set. *Proceedings of IJCNN-89, June 18-22, 1989, Washington, D.C.* vol. I, 67-75.

[10] Eliashberg, V. (1990). Molecular dynamics of short-term memory. *Mathematical and Computer modeling in Science and Technology.* vol. 14, 295-299.

[11] Eliashberg, V. (2005). Ensembles of membrane proteins as statistical mixed-signal computers. *Proceedings, IJCNN 2005.*

[12] Gerstner, W., Kistler, M. (2002). Spiking Neuron Models. Single Neurons, Populations, Plasticity. *Cambridge University Press.*

[13] Hille, B. (2001). Ion Channels of Excitable Membranes. *Sinauer Associates, Inc.*

[14] Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of ion currents and its applications to conduction and excitation in nerve membranes. *J. Physiol. (Lond.)*, 117:500-544.

[15] Feldman, J.A. (2006). From Molecule to Metaphor. A Neural Theory of Language. *A Bradford Book, the MIT Press.*

[16] Gold, E.M. (1967). Language identification in the limit. *Information and Control, 10:447-474.*

[17] Kandel, E.,R. (2006). In Search of Memory. The Emergence of a New Science of Mind. *W.W. Norton and Co.*

[18] Kosko, B. (1992). Neural Networks and Fuzzy Systems. A Dynamical Systems Approach to Machine Intelligence. *Prentice Hall, Inc.*

[19] Minsky, M., Papert, S. (1969). Perceptrons. An Introduction to Computational Geometry, *The MIT Press.*

[20] Minsky, M.L. (1967). Computation: Finite and Infinite Machines, *Prentice-Hall, Inc.*

[21] Nauta, W., Feirtag, M. (1986). Fundamental Neuroanatomy, *W.H. Freeman and Co..*

[22] Nagel, E., and Newman J.R. (1958). Gödel's Proof, *New York University Press.*

[23] Turing, A.M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Society*, ser. 2, 42

[24] Vvedensky, N.E. (1901). Excitation, inhibition, and narcosis. *In Complete collection of works. USSR, 1953.*

[25] Wooldridge, D.E. (1963). The Machinery of the Brain. *The McGraw-Hill Book Company, Inc.*

[26] Zopf, G.W. (1961). Attitude and Context. In "Principles of Self–organization". *Pergamon Press*, 325-346.

[27] Kennedy, W.G., and De Jong, K.A. (2003). Characteristics of long-term learning in Soar and its applications to the utility problem. *Proceedings of the Twentieth International Conference on Machine Learning, pp. 337-344.*

[28] Kennedy, W.G., and Trafton, J.G (2006). Long-term symbolic learning in Soar and ACT-R. *Navy Center for Applied Research in Artificial Intelligence. ACT-R group Web publications, http://act-r.psy.cmu.edu/publications/index.php?subtopic=55*

[29] Minton, S. (1990). Quantitative results concerning the utility of explanation-based learning, *Artificial Intelligence, 42, pp. 363-392.*

[30] Dusan, S., Rabiner, L.R. (2005). Can automatic speech recognition learn more from human speech recognition. *Trends in speech technology. Romanian Academic Publisher.*

[31] Pellom, B., Hacioğlu, K. (2001). SONIC. *Technical Report TR-CSLR-2001-01. Center for Spoken Language Research, Univercity of Colorado, Boulder.*

[32] Eliashberg, V., Eliashberg, Y. (2007). The Mathematics of the Brain. Proposal for DARPA/DSO SOL, DARPA Mathematical Challenges, BAA 07-68. Mathematical Challenge One. Grant, Award No. FA9550-08-1-0129.

[33] Eliashberg, V. (2009). A nonclassical symbolic theory of working memory, mental computations, and mental set. *In press. Available from http://arxiv.org/abs/0901.1152 and www.brain0.com.*

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
27-05-2009	Final Performance Report	15-03-2008 -- 14-03-2009

4. TITLE AND SUBTITLE
The Mathematics of the Brain

5a. CONTRACT NUMBER
FA9550-08-1-0129

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Eliashberg, Victor, Dr., Eliashberg, Yakov, Dr.

5d. PROJECT NUMBER

5e. TASK NUMBER
1,2,3,4, and 6,7,8,9,10

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Avel Electronics
3450 Murdoch Court
Palo Alto, CA 94306-3633

8. PERFORMING ORGANIZATION REPORT NUMBER
AE-MOB-001

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
DCMA Northern California
P.O.BOX 232
700 Roth Road, Bldg. 330
French Camp, CA 95231-0232

Defense Sciences Office, DARPA
AF Office of Scientific Research

10. SPONSOR/MONITOR'S ACRONYM(S)
S0507A, DARPA DSO, AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-OSR-VA-TR-2012-0559

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The project develops a formalism for bridging the gap between symbolic and dynamic levels of the mathematical theory of the brain. The formalism is based on the hypotheses that the neocortex processes symbolic information in a nonclassical way. Instead of manipulating data in a read/write memory, as the classical symbolic systems do, the neocortex changes the states of dynamic memory representing different temporary attributes of data stored in a long-term memory. This nonclassical symbolic/dynamic computational paradigm is called the concept of E-machine. Intuitively, an E-machine manipulates characteristic functions over the sets of memory addresses of data rather than addresses and data themselves. The main results include: (1) demonstrating the Turing universality of the E-machine paradigm, (2) showing how an E-machine can learn to simulate a read/write working memory without actually re-writing data, (3) demonstrating how the E-machine paradigm can be efficiently implemented in biologically-realistic neural networks with temporal modulation. Several computer simulations supporting the theoretical results were performed.

15. SUBJECT TERMS
Cognitive modeling, neural networks, symbolic/dynamic integration, context-sensitive associative memory, working memory

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Victor Eliashberg
	U	U	SAR	44	19b. TELEPHONE NUMBER: (650) 493-6038

Standard Form 298